Traitement STAP en environnement hétérogène. Application à la détection radar et implémentation sur GPU

Jean-François Degurse

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE : Sciences et Technologie de l'Information, des Télécommunications et des Systèmes

ONERA - Département Électromagnétisme et Radar
Laboratoire des Signaux et des Systèmes (SUPÉLEC-CNRS-UPS) - UMR 8506

DISCIPLINE : Physique

THÈSE DE DOCTORAT

soutenue le 15/01/2014

par

Jean-François DEGURSE

Traitement STAP en environnement hétérogène. Application à la détection radar et implémentation sur GPU.

Directeur de thèse :	Sylvie MARCOS	Directeur de Recherche (L2S SUPÉLEC)
Encadrants :	Laurent SAVY	Ingénieur (ONERA)
	Jean-Philippe MOLINIÉ	Ingénieur (ONERA)
Composition du jury :		
Rapporteurs :	Olivier BESSON	Directeur de Recherche (Université de Toulouse - ISAE)
	Philippe FORSTER	Professeur (Université Paris Ouest - SATIE, ENS Cachan)
Examinateurs :	François LE CHEVALIER	Professeur (Delft University of Technology - THALES)
	Jean-Luc MILIN	Dr-Ingénieur (Direction Générale de l'Armement)
	Marc MONTECOT	Ingénieur (THALES Airborne Systems)
	Sylvie MARCOS	Directeur de Recherche (L2S SUPÉLEC)
	Laurent SAVY	Ingénieur (ONERA)
	Philippe ZARKA	Directeur de Recherche (Observatoire de Paris)
Membre invité :	Frédéric BARBARESCO	Dr-Ingénieur (THALES Air Systems)

Remerciements

Je voudrais tout d'abord remercier l'ONERA, le Laboratoire des Signaux et des Systèmes ainsi que la DGA pour m'avoir offert l'opportunité d'effectuer cette thèse. Je remercie très sincèrement mon directeur de thèse Sylvie MARCOS, pour toute son aide et sa disponibilité. Sa rigueur, ses connaissances et son amabilité ont été pour moi sources de motivation et m'ont permis d'avancer tout au long de ma thèse. Je remercie également très sincèrement mon encadrant à l'ONERA, Laurent SAVY, qui a consacré beaucoup de son temps à ma thèse, son dynamisme et ses connaissances m'ont permis de progresser rapidement dans mes travaux. Je me considère vraiment chanceux de vous avoir eu comme encadrants et c'est grâce à vous que j'ai pu mener cette thèse à terme dans de bonnes conditions.

Je remercie le Département Electromagnétisme et Radar (DEMR) de l'ONERA, par l'intermédiaire de son directeur Jean-Marc BOUTRY, pour m'avoir accueilli et m'avoir donné les moyens d'effectuer ma thèse. Je remercie également, Elisabeth BERTHEAU et François RICCI du secrétariat du DEMR que j'ai beaucoup sollicité avec mes interrogations administratives ainsi que Hubert CANTALLOUBE pour toute l'aide informatique qu'il m'a apporté. Évidement, j'ai une pensée particulière pour les membres de l'équipe RBF du DEMR dans laquelle j'ai été durant cette thèse et notamment Daniel LE COZ sans qui je n'aurais pas eu l'occasion de faire cette thèse, Juanito MUNOZ avec qui j'ai pris un grand plaisir à discuter informatique et qui a toujours su répondre à mes questions. Je remercie très chaleureusement et amicalement Jean-Philippe MOLINIE, qui m'a d'abord encadré durant mon stage qui a précédé la thèse, puis qui a co-encadré ma thèse et que j'ai beaucoup (trop !) sollicité mais qui a toujours pris du temps pour me répondre. Ce fut vraiment un plaisir de travailler ensemble.

J'ai aussi une pensée pour les thésards et apprentis que j'ai rencontré à l'ONERA : Thomas, Marc, Luca, Nicolas, Corinna, Damien-Barthélémy, Rata, Chirstelle, Oana et Mathieu, ainsi que pour les personnels et thésards du L2S à Supélec : Nabil, Thang, Elsa, Pierre, Thomas, Alex, Nicolas, Achal, Benjamin, Anna, Vineeth, François, Mael, Amadou, José et Nicolas avec qui l'ambiance a été très agréable tout au long de ces trois années.

Enfin je tiens à remercier mes proches pour leur soutien au quotidien ainsi que toutes les personnes qui se sont déplacés pour ma soutenance de thèse.

Paris, le 4 février 2014.

« It doesn't matter how beautiful your theory is, it doesn't matter how smart you are.
If it doesn't agree with experiment, it's wrong.
In that simple statement is the key to science. »
Richard Feynman

Table des matières

Annexe C
STAP sur GPU

Table des figures

Introduction générale

Les traitements STAP sont des traitements qui exploitent conjointement les deux dimensions spatiale et temporelle des signaux reçus sur un réseau d'antennes, contrairement au traitement d'antenne classique qui n'exploite que la dimension spatiale, pour leur filtrage et séparation. Cette structure de traitement permet de tirer parti des propriétés spécifiques bidimensionnelles spatio-temporelles, ou dans le domaine dual, angle-fréquence, des signaux reçus. Cela s'avère particulièrement intéressant notamment dans le cas d'une propriété de couplage angle-fréquence des signaux reçus, où, si les signaux sont étendus dans les deux espaces pris séparément, ils n'occupent cependant qu'une dimension dans l'espace 2D. Leur filtrage devient alors possible par traitements STAP, alors qu'il ne l'était pas par traitement mono-dimensionnel spatial ou temporel.

Cette problématique se rencontre en particulier dans le cadre du filtrage des échos reçus par un radar aéroporté en provenance du sol, pour lesquels il existe un lien direct entre direction d'arrivée et fréquence Doppler. Ces échos de sol, ou fouillis, sont classiquement filtrés en radar par un filtrage spatial suivi d'un traitement Doppler (filtrage fréquentiel). Ces échos, étendus dans les deux domaines spatial et fréquentiel, ne sont ainsi qu'imparfaitement filtrés, et leurs résidus limitent encore fortement les performances en détection.

Dans ce contexte, l'emploi de traitements STAP est donc d'un apport majeur, et leur implantation dans des applications opérationnelles est maintenant envisageable à court terme avec l'arrivée d'antennes actives à réception multi-voies dans des produits industriels.

Les deux applications principales sont la détection, par le radar de pointe avant d'un avion de combat, des cibles terrestres mobiles lentes en mode Air-Sol (cible en compétition avec les échos de fouillis entrant par le lobe principal de l'antenne), et l'amélioration de la détection des cibles aériennes sur les zones de la carte radar distance-Doppler polluées par le fouillis en mode Air-Air (cible en compétition avec les échos de fouillis vus à travers les lobes secondaires de l'antenne).

Cependant, si les principes des traitements STAP sont maintenant bien acquis, leur mise en œuvre pratique face à un environnement réel se heurte à des points durs non encore résolus dans le contexte du radar de pointe avant d'un avion de combat.

Le premier verrou, adressé par la thèse dans une première partie, est d'ordre théorique, et consiste en la définition de procédures d'estimation de la matrice de covariance du fouillis sur la base d'une sélection des données d'apprentissage représentatives, dans un contexte à la fois de fouillis non homogène et de densité parfois importante des cibles d'intérêts. La plupart des travaux publiés dans la littérature ouverte dans le domaine du STAP adressent en effet le contexte applicatif de radars de surveillance longue portée à visée latérale et des réseaux d'antennes linéaires uniformes, associé à des formes d'onde non ambiguës en distance. Ces spécificités ne correspondent pas au cas généralement rencontré sur un avion de combat, ce qui se traduit par une inhomogénéité spatiale forte du fouillis (due notamment à la présence d'ambiguïtés distance) qui rend caduque l'exploitation d'algorithmes classiques basés sur l'exploitation d'un volume important de données d'apprentissage ne contenant pas le signal d'intérêt. En effet, les algorithmes classiques de détection d'une cible dans une case sous test utilisent des cases distance voisines pour estimer la matrice de covariance du fouillis, l'hypothèse étant que les signaux reçus sur les

cases distance adjacentes sont indépendants et distribués de manière identique au fouillis de la case sous test, éventuellement à une variable de texture près. En particulier, les modèles SIRV, aujourd'hui les plus élaborés, qui introduisent une variable de texture, sont mal adaptés à la représentation du fouillis en présence d'ambiguïtés distance (la composante gaussienne du fouillis est elle même rapidement variable en distance).

L'objectif de la première phase de la thèse est donc de proposer des méthodes de détection de cibles en présence d'un fouillis fortement hétérogène et/ou d'environnement dense en cibles d'intérêts. Pour résoudre ce problème deux voies seront explorées. La première voie consistera à définir des traitements STAP sans données d'apprentissage, pour les cas de fouillis très hétérogènes ou d'environnement très dense en cibles. La seconde voie consistera à rechercher des critères de sélection adaptatifs des cases distance qui serviront à estimer la matrice de covariance du fouillis. Parallèlement, nous rechercherons des tests de détection plus robustes correspondant à des classes de signaux plus larges.

Le second verrou est d'ordre technologique, et réside dans l'implémentation physique des algorithmes, lié à la grande charge de calcul nécessaire. Ce point, crucial en aéroporté, sera exploré dans la deuxième partie de la thèse, avec l'analyse de la faisabilité d'une implémentation sur GPU (*Graphic Processor Unit*) des algorithmes les plus prometteurs. Les processeurs GPU offrent en effet potentiellement des puissances de calcul importantes pour un encombrement et un coût réduits, sous réserve d'une architecture algorithmique pouvant s'adapter à l'aspect parallélisation nécessaire aux GPU. Tous les algorithmes ne pourront donc pas tirer partie de la puissance du GPU, et cet aspect est à prendre en compte dès la phase de design théorique de l'algorithmie.

Première partie

Contexte et état de l'art

Chapitre 1

Contexte : le radar aéroporté

Sommaire

Contrairement aux traitements d'antenne classiques qui n'exploitent que la dimension spatiale des signaux reçus pour leurs filtrages et séparations, les traitements STAP (*Space Time Adaptive Processing*) sont des traitements qui exploitent conjointement les deux dimensions (spatiale et temporelle) des signaux reçus sur un réseau d'antennes. Cette structure de traitement permet de tirer parti des propriétés spécifiques spatio-temporelles, ou dans le domaine dual, angle-fréquences, des signaux reçus. Cela s'avère particulièrement intéressant notamment dans le cas d'une propriété de couplage angle-fréquence des signaux reçus, où, si les signaux sont étendus dans les deux espaces pris séparément, ils n'occupent cependant qu'une dimension dans l'espace 2D. Leur filtrage/séparation devient alors possible par traitements STAP, alors qu'elle ne l'était pas par traitement mono-dimensionnel spatial ou temporel. Cette problématique se rencontre en particulier dans le cadre du filtrage des échos reçus par un radar aéroporté en provenance du sol pour lesquels il existe un lien direct entre direction d'arrivée et fréquence Doppler. Ces échos de sol, ou fouillis, sont classiquement filtrés en radar par un traitement spatial (filtrage spatial par le lobe d'antenne) suivi d'un traitement Doppler (filtrage fréquentiel). Ces échos, étendus dans les deux domaines spatial et fréquentiel, ne sont ainsi qu'imparfaitement filtrés, et leurs résidus limitent encore fortement les performances en détection.

Dans une première section, nous présenterons la problématique générale de la détection en radar aéroporté. Nous détaillerons ensuite le contexte et la problématique du traitement STAP. Nous présenterons ensuite les principes généraux du traitement STAP dans une configuration d'antenne à visée latérale ainsi qu'une antenne à visée frontale.

1.1 Généralités

En radar, le traitement du signal se compose des tâches de détection et d'estimation de la position des cibles. La tâche de détection demande le plus gros effort par rapport à la localisation qui n'est faite que pour les cibles détectées. Celles-ci sont peu nombreuses en comparaison du nombre de cases distance-Doppler-angle à tester [1]. L'objectif final du traitement radar est d'assurer une bonne probabilité de détection P_d tout en maintenant une très faible probabilité de fausse alarme P_{fa}.

La plupart des détecteurs à taux de fausse alarme constant (CFAR) compare la statistique de la case distance-Doppler-angle qui est testée à celle des données des cases voisines, appelées également données secondaires. Classiquement, les détecteurs comparent le niveau de puissance de la case sous test au niveau moyen reçu sur un groupe de cases voisines. La détection est déclarée si le niveau de la case sous test dépasse d'un certain seuil le niveau dans les cases secondaires. Ce seuil est choisi pour assurer une probabilité de détection suffisante, tout en maintenant une faible probabilité de fausse alarme. Les probabilités de détection et de fausse alarme sont affectées de manière opposée par la présence d'interférence (échos de sol ou brouilleurs). Tout l'enjeu du traitement radar est de présenter une grande robustesse vis-à-vis de ces signaux parasites, en maintenant une P_d élevée tout en maitrisant la P_{fa}. Pour les cibles détectées, le traitement radar doit également conserver la possibilité de mesures distance, Doppler, et angle du signal.

Les antennes à balayage électronique actives (voir Annexe A.2) procurent plusieurs voies en réception spatiales et offrent donc la possibilité de mise en œuvre de traitements spatio-temporels adaptatifs (STAP) pour la suppression de fouillis et du brouillage, en exploitant les propriétés spécifiques des deux dimensions spatiales et temporelles (ou angle-fréquence). Ces traitements améliorent sensiblement la détection par rapport au cas mono-voie. Ils sont par ailleurs adaptatifs, c'est à dire qu'ils s'adaptent aux exigences et contraintes de l'environnement du radar, et il n'est pas envisageable de recourir à un traitement non-adaptatif pour les raisons suivantes : tout d'abord la configuration et l'environnement ne sont pas connus avec précision (mouvement du porteur, trajectoire, reliefs, etc...), ensuite les défauts de calibrage entre les voies de réception induisent un biais qu'il n'est pas possible de maitriser alors que le niveau du signal des interférences à supprimer est très supérieur à celui du signal d'intérêt (2 à 4 ordres de grandeur).

1.2 Antenne à visée latérale

1.2.1 Traitements STAP

Dans cette configuration, l'antenne est à implantation latérale (voir Figure 1.1), et la forme d'onde radar est typiquement BFR (Basse fréquence de Récurrence), évitant ainsi des ambiguïtés distance. La problématique est la détection des cibles lentes en compétition avec les échos de sol, appelés fouillis en provenance du lobe principal de l'antenne et vus à des vitesses comparables à celles des cibles recherchées.

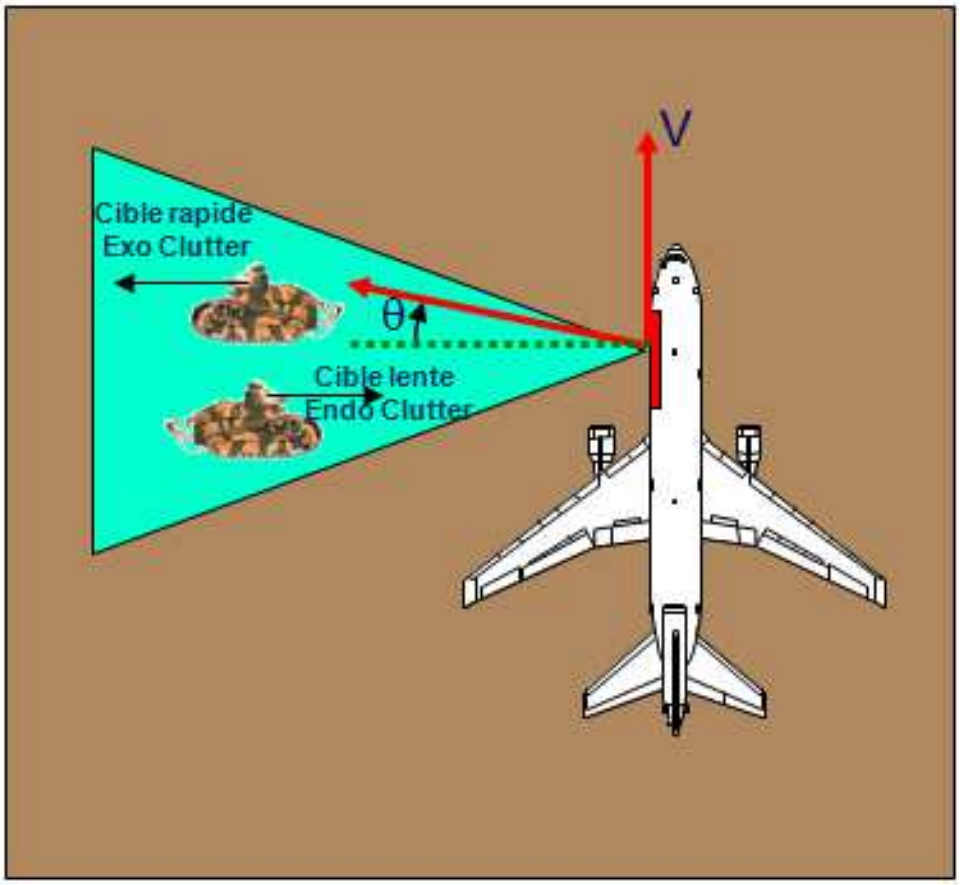


FIGURE 1.1 – Géométrie de la configuration Radar à antenne latérale

Dans cette configuration, la fréquence Doppler F_d d'un écho de fouillis est directement reliée à son angle d'arrivée θ dans le repère antenne (voir Figure 1.2) par la relation fondamentale pour une antenne latérale

$$F_d = \frac{2V}{\lambda} \sin\theta \tag{1.1}$$

ce qui conduit à un étalement Doppler des échos de sol vu par le lobe principal de

$$\Delta F_d = \pm\frac{2V}{\lambda} \sin\theta_{3dB} \simeq \pm\frac{2V}{\lambda}\theta_{3dB} \tag{1.2}$$

où θ_{3dB} est l'ouverture à 3dB du lobe d'antenne.

Un traitement spatio-temporel bidimensionnel peut tirer parti de la relation (1.1), qui indique que dans le plan angle-Doppler, le fouillis s'étend le long d'une droite, et n'occupe donc qu'un espace 1D dans le plan 2D (Figure 1.3).

La mise en œuvre d'un filtre bidimensionnel spatio-temporel opérant le long de la droite de fouillis permet alors la réjection du fouillis tout en conservant les cibles endo clutter, alors qu'aucun des deux traitements monodimensionnels d'élimination du fouillis, filtrage spatial ou filtrage temporel (appelé aussi classiquement filtrage Doppler) ne permet de conserver la cible endo-clutter.

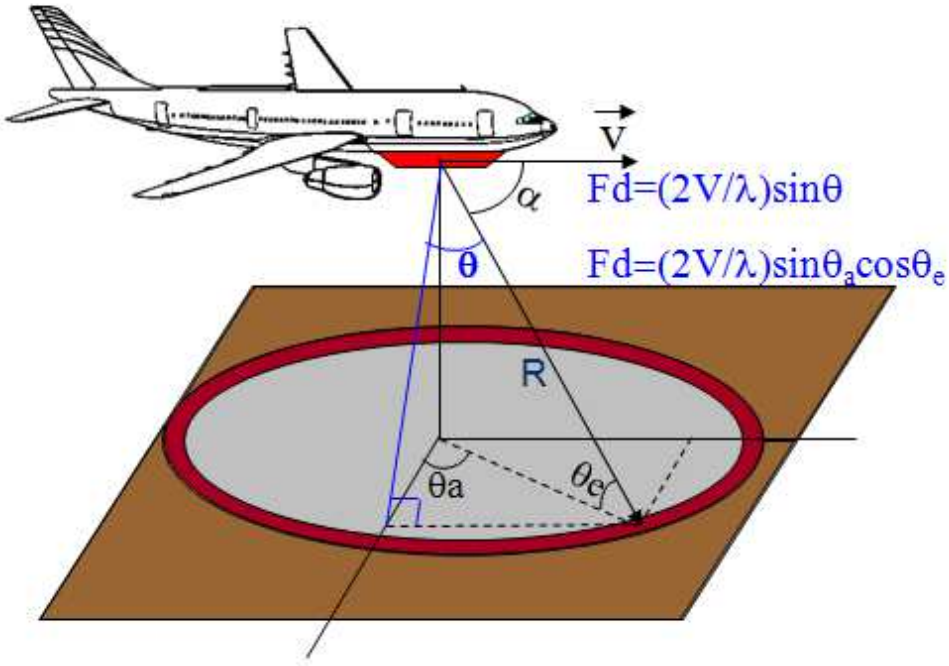


FIGURE 1.2 – Antenne à visée latérale, θ_a et θ_e sont les angles d'azimut et d'élévation.

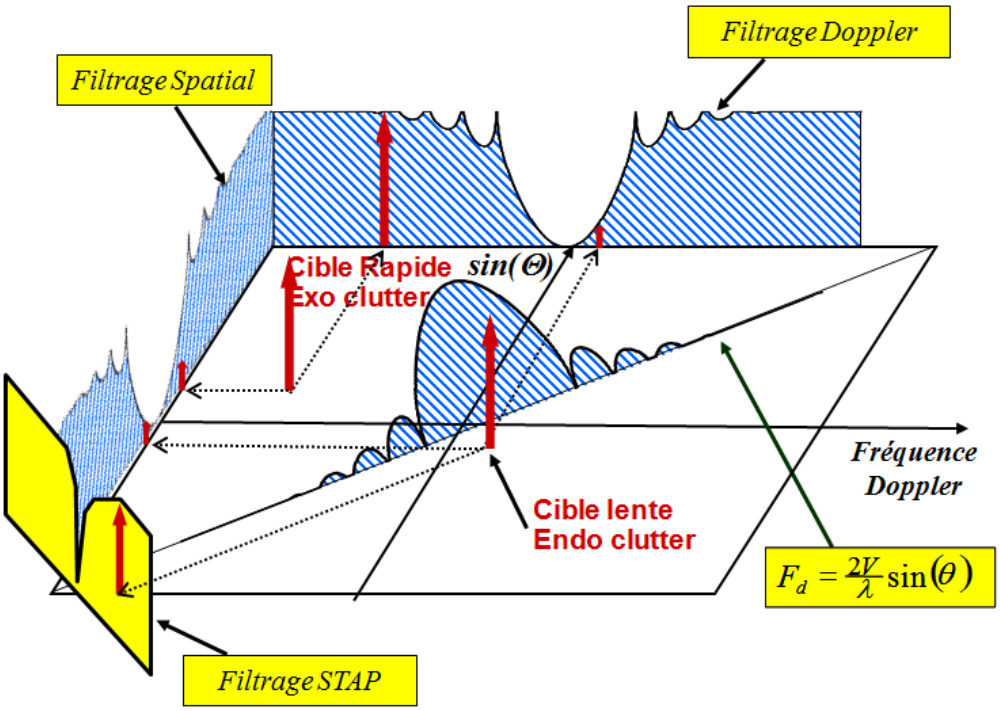


FIGURE 1.3 – Filtrage STAP dans le plan angle-Doppler pour la configuration radar à antenne latérale

1.2.2 Traitement STAP : point de vue fréquentiel

L'analyse précédente mène au principe des traitements STAP post-doppler. En effet, il est évident d'après la Figure 1.3 que l'élimination du fouillis peut être réalisée par un filtre spatial mettant à zéro les signaux se trouvant sur la droite du fouillis dont la position dépend de la fréquence Doppler. Cette approche est appelée *Factored* STAP dans la littérature. On suppose, dans la suite, une antenne parfaitement alignée sur le vecteur vitesse et constituée de deux sous réseaux identiques espacés d'une distance D (voir Figure 1.4), recevant respectivement les signaux $s_0(t)$ et $s_1(t)$.

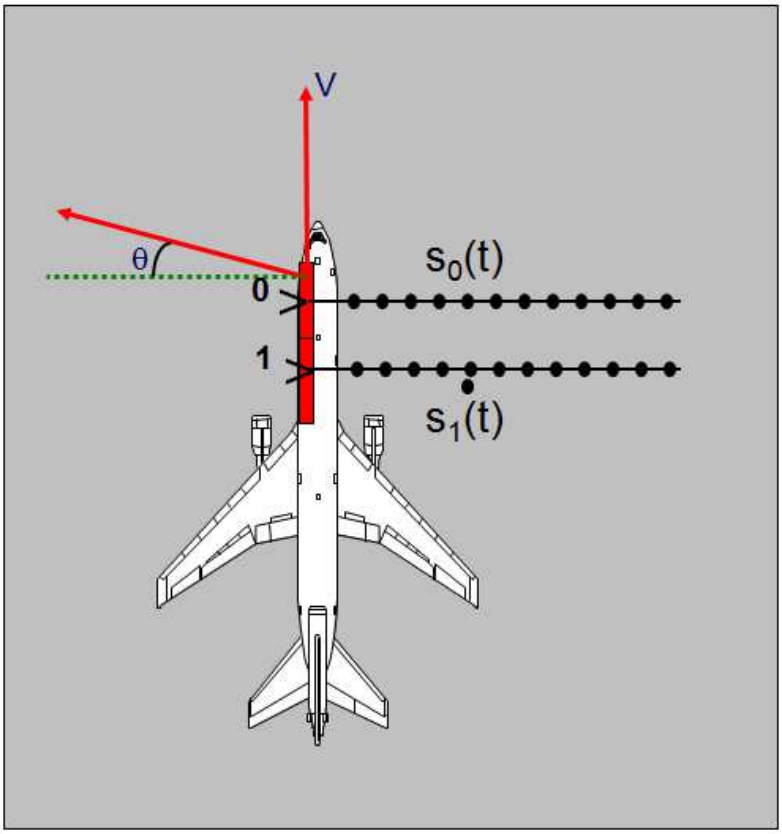


FIGURE 1.4 – Antenne à visée latérale constituée de 2 voies de réception

Une expression analytique du filtrage STAP post-Doppler peut alors être obtenue

$$X_{\text{post-Doppler}}(F_d) = \widetilde{s_1}(F_d)\mathrm{e}^{2j\pi\frac{D}{\lambda}\sin\theta_c} - \widetilde{s_0}(F_d) \tag{1.3}$$

avec λ la longueur d'onde, $\widetilde{s_1}(F_d)$ et $\widetilde{s_0}(F_d)$ les signaux en fréquence des sous-réseaux 0 et 1, θ l'angle d'arrivée du fouillis à la fréquence Doppler F_d et D la distance entre les deux capteurs. Comme nous l'avons vu dans l'équation (1.1), nous avons la relation

$$\sin\theta = F_d\frac{\lambda}{2V}$$

et le filtrage s'écrit

$$X_{\text{post-Doppler}}(F_d) = \widetilde{s_1}(F_d)\mathrm{e}^{2j\pi\frac{D}{\lambda}F_d\frac{\lambda}{2V}} - \widetilde{s_0}(F_d) \tag{1.4}$$

L'équation (1.4) montre qu'il s'agit d'un filtrage spatial qui dépend de la fréquence, il est donc spatio-fréquentiel. Pour une cible qui se trouve au centre du lobe d'antenne, nous avons $\widetilde{s_0}(F_d) = \widetilde{s_1}(F_d) = A$ et

$$F_d = \frac{2v_r}{\lambda}$$

avec v_r la vitesse radiale du porteur, et une ouverture du lobe d'antenne

$$2\theta_{3dB} = \frac{\lambda}{D} \iff \frac{D}{\lambda} = \frac{1}{2\theta_{3dB}}$$

où θ_{3dB} est est la demi-ouverture azimutale du lobe à 3dB.

Nous pouvons déduire l'allure du filtre STAP en ré-écrivant l'équation (1.4) :

$$X_{\text{post-Doppler}}(F_d) = A\mathrm{e}^{j\pi\frac{v_r D}{V\lambda}}.2j\sin\left(\pi\frac{v_r D}{V\lambda}\right) = A\mathrm{e}^{j\pi\frac{v_r D}{V\lambda}}.2j\sin\left(\frac{\pi v_r}{2V\theta_{3dB}}\right) \tag{1.5}$$

L'équation (1.5) mène à la forme du filtre STAP de la Figure 1.5.

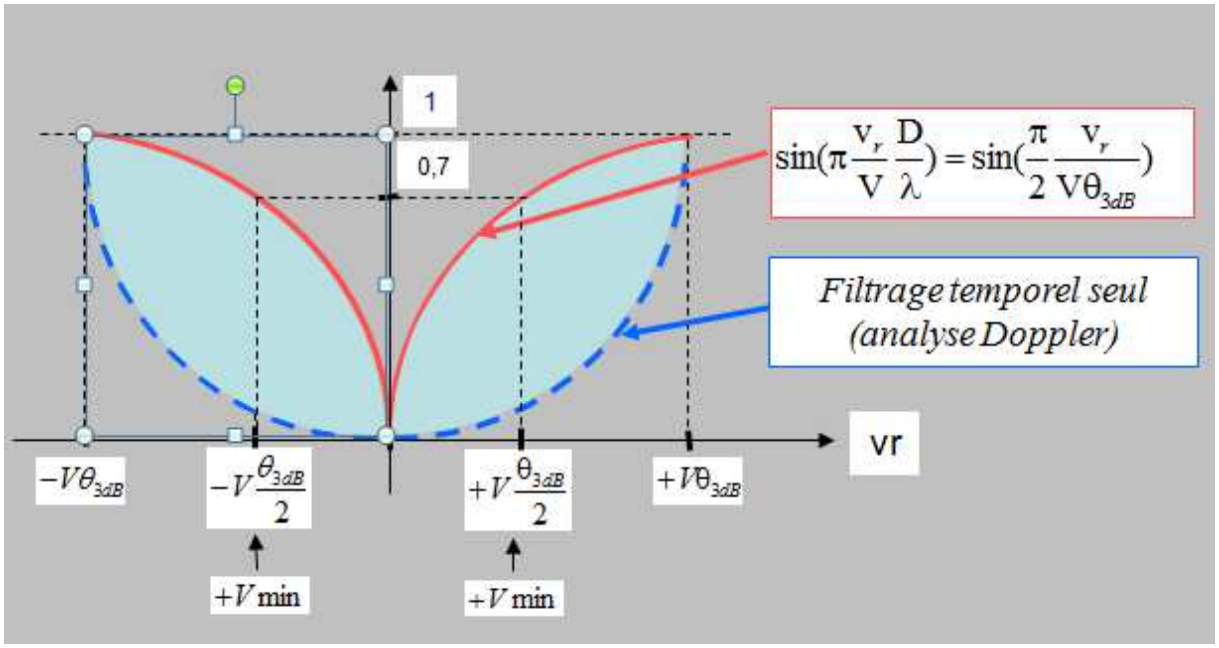


FIGURE 1.5 – Forme du filtre spatio-temporel (rouge) et du filtre temporel seul (bleu pointillés) pour une antenne radar à visée latérale.

La Figure 1.5 montre l'apport d'un traitement STAP par rapport à un traitement Doppler seul sur une configuration radar à antenne latérale. Nous voyons clairement le gain qu'amène le traitement STAP aux faibles vitesses (surface bleue ciel). Ainsi la vitesse minimale de détection des cibles est bien plus faible avec un traitement STAP par rapport à l'analyse Doppler classique.

1.2.3 Traitement STAP : point de vue temporel

Le traitement STAP pré-Doppler (dual du traitement post-Doppler précédent) se déduit immédiatement par transformée de Fourier inverse de l'Eq. (1.5) par rapport à F_d :

$$X_{\text{pré-Doppler}}(F_d) = s_1(t + \frac{D}{2V}) - s_0(t) \tag{1.6}$$

qui peut aussi s'écrire de la façon suivante

$$X_{\text{pré-Doppler}}(F_d) = A\mathrm{e}^{2j\pi F_d t}\mathrm{e}^{j\pi \frac{v_r D}{V\lambda}}.2j\sin\left(\pi\frac{v_r D}{V\lambda}\right) \tag{1.7}$$

Le traitement STAP pré-Doppler consiste donc à soustraire les signaux issus des deux sous réseaux à des instants décalés de $D/2V$ (voir Figure 1.6). Ce traitement, précurseur des traitements STAP, est référencé dans la littérature sous le nom de traitement DPCA (*Displaced Phase Center Antenna*)[2]. Il peut également se déduire de manière intuitive en remarquant que le centre de phase de l'ensemble émetteur-récepteur du sous-réseau 1 aura pris physiquement la place du centre de phase de l'ensemble émetteur-récepteur du sous-réseau 1 après une durée de $\Delta t = D/2V$, et verra donc exactement le même fouillis à un instant $+D/2V$ plus tard.

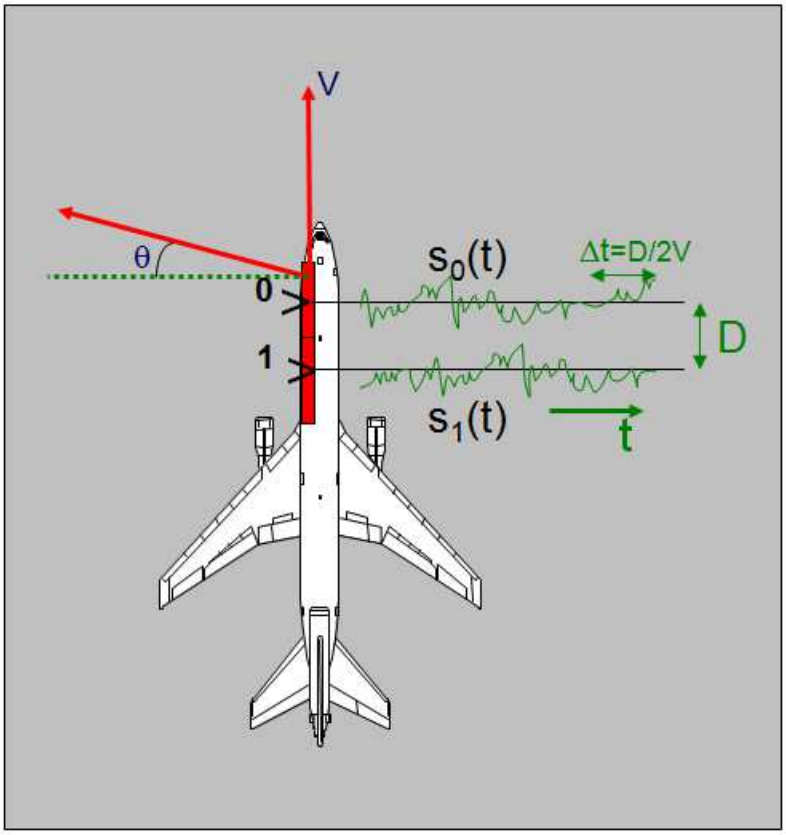


FIGURE 1.6 – Traitement STAP pré-Doppler sur une Antenne à visée latérale

1.3 Antenne à implantation frontale

La configuration radar à antenne frontale est typiquement celle de radars implantés sur les avion d'armes. La forme d'onde radar est typiquement MFR (Moyenne fréquence de Récurrence), avec présence d'ambiguïtés distance. La problématique est, en plus, la détection des cibles aériennes (mode Air-Air) en compétition avec les échos de fouillis en provenance des lobes secondaires de l'antenne (mode veille) ou en provenance du lobe principal (mode poursuite).

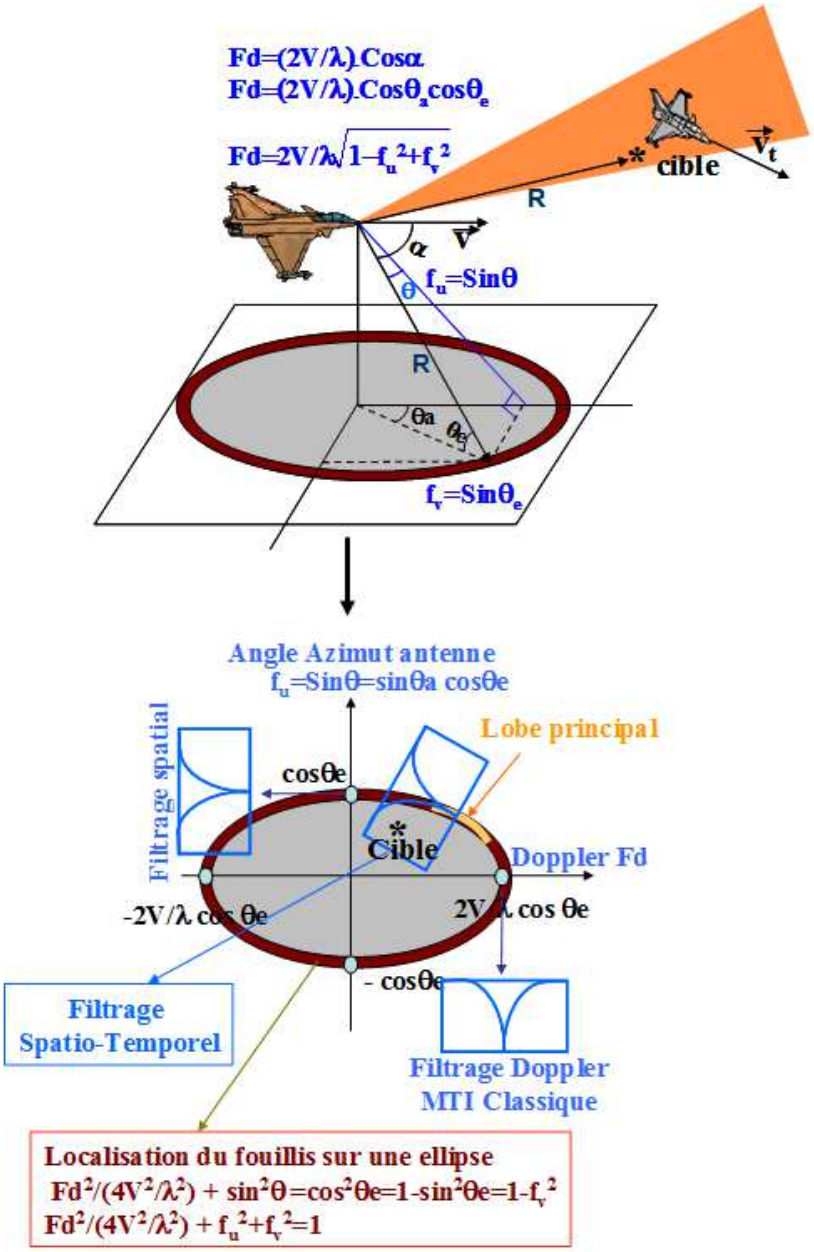


FIGURE 1.7 – Localisation du fouillis dans le plan Angle azimut-Doppler pour la configuration radar à antenne frontale

L'implantation frontale de l'antenne conduit à une répartition du fouillis dans le plan angle-Doppler plus complexe que précédemment. Pour une case distance non ambiguë, celui-ci se concentre en effet sur une ellipse (Figure 1.7). La taille de cette ellipse dépend de l'angle d'élévation θ_e sur la Figure 1.7, sous lequel est vu le fouillis, et donc de la distance R considérée. Pour appréhender correctement la problématique STAP dans cette configuration, il faut considérer non plus seulement le plan angle-azimut-Doppler mais l'espace tridimensionnel angle-azimut-angle d'élévation-Doppler ou plus précisément l'espace (f_u, f_v, F_d) des cosinus directeurs-Doppler.

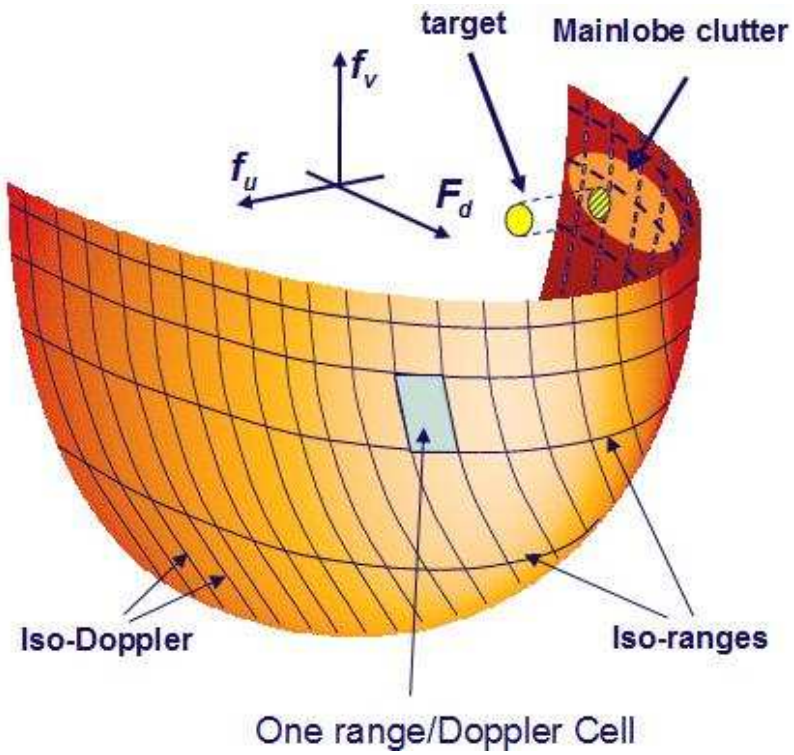


FIGURE 1.8 – Localisation du fouillis dans l'espace (f_u, f_v, F_d). Les lignes horizontales sont les lignes iso-distance. Les lignes verticales représentent les lignes iso-Doppler

La localisation du fouillis dans cet espace est le demi-ellipsoïde de la Figure 1.8, dont la forme limite est un demi-plan dans le cas de la configuration latérale [3]. La coupe à un angle d'élévation donné, c'est à dire une distance donnée, sur la Figure 1.8 correspond à la droite indiquée Figure 1.3 dans l'espace angle azimut-Doppler. La configuration en antenne frontale génère donc une dépendance en distance de la loi reliant l'angle d'arrivée et la fréquence Doppler des échos de fouillis. A cette non stationnarité en distance du fouillis, se rajoute le plus souvent la présence d'ambiguïtés distance, et dans tous les cas des hétérogénéités intrinsèques à la nature variable du fouillis observé. Cet aspect n'est pas sans conséquence pour l'apprentissage des coefficients adaptatifs du filtre STAP, qui doit donc être limité sur une petite zone distance.

1.4 Synthèse

Dans cet partie, nous avons énoncé les principes généraux ainsi que l'intérêt des traitements STAP dans le contexte d'un radar aéroporté. Nous avons montré qu'en théorie, ces traitements permettent d'améliorer sensiblement la détection des cibles par rapport à un traitement Doppler classique. Dans le chapitre suivant, nous allons décrire plus en détail les principes de ces traitements et montrer qu'en pratique leur utilisation peut être limitée à cause de l'hétérogénéité de l'environnement. Nous dresserons enfin un état de l'art des travaux liés à cette problématique.

Chapitre 2

Traitement STAP : Notations et état de l'art

Sommaire

Dans ce chapitre, nous étudions les principes de traitement spatio-temporel d'un radar muni d'une antenne comportant plusieurs voies de réception. Nous définissons les notations utilisées dans la suite de la thèse ainsi le traitement STAP de référence. Après avoir présenté ce que sont les milieux hétérogènes et leur problématique dans le cadre d'un traitement STAP, nous dressons un état de l'art des méthodes existantes permettant de résoudre une partie des limitations liées aux environnements hétérogènes. Enfin, nous concluons en plaçant nos travaux dans ce contexte.

2.1 Traitement STAP : principe

Considérons un radar muni d'une antenne constituée de plusieurs capteurs tel que représenté sur la Figure 2.1. A l'émission, une antenne transmet des impulsions successives (période de répétition Tr), et en réception, un réseau d'antennes, considéré linéaire et uniforme, reçoit les signaux.

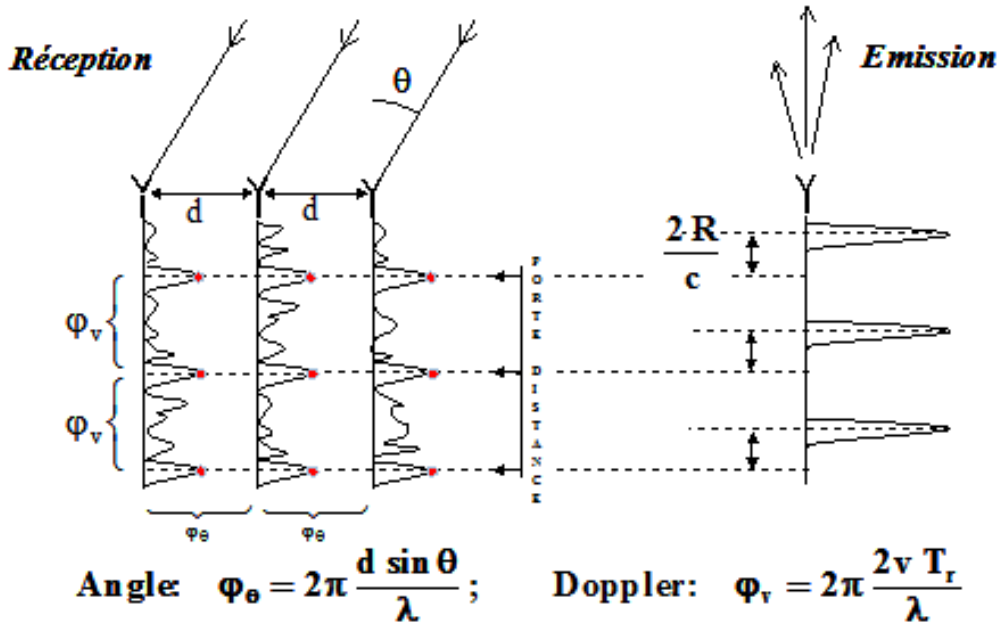


FIGURE 2.1 – Antenne de réception multi-canaux

Les signaux provenant d'une cible se trouvant à la distance R sont indiqués par un point rouge sur la Figure 2.1. En l'absence de bruit, les échantillons de ce signal seraient caractérisés par le déphasage ϕ_v d'impulsion à impulsion, et ϕ_θ de capteur à capteur. Le principe du traitement standard est de sommer ces échantillons, après les avoir remis en phase pour une hypothèse angle-vitesse (v, θ) et d'effectuer ce traitement pour toutes les hypothèses souhaitées. C'est le principe de la formation de faisceaux par le calcul (*digital beamforming*) et du traitement Doppler. Des traitements de ce type peuvent avoir de nombreuses applications en radioastronomie [4][5], en navigation satellite (GNSS) [6], en radiocommunications ou pour des systèmes sonar [7].

2.1.1 Cas sans interférence : bruit thermique seul

Soit un réseau de réception linéaire composé de N capteurs espacés uniformément d'une distance D égale à une demi longueur d'onde. Le radar émet un train de M impulsions (dimension temporelle d'un bloc STAP pré-Doppler) à une période de répétition (PRF) T_r. Nous adoptons le modèle de signal défini dans [8]. Le vecteur $\mathbf{s_s}(v_r, \theta)$ des signaux reçus d'une cible d'amplitude unité dans la direction θ et de vitesse radiale v_r

s'écrit

$$\mathbf{s_s}(\theta, v_r) = \mathbf{b}(v_r) \otimes \mathbf{a}(\theta) \tag{2.1}$$

où $\mathbf{a}(\theta)$ et $\mathbf{b}(v_r)$ sont respectivement les vecteurs directeurs spatial de dimension N et temporel de dimension M et $\otimes$ est le produit de Kronecker. Le vecteur $\mathbf{a}(\theta)$ est défini par

$$\mathbf{a}(\theta) = \left[1 \ \exp\left(\frac{j2\pi D \sin\theta}{\lambda}\right) \ \ldots \ \exp\left(\frac{j2\pi D(N-1)\sin\theta}{\lambda}\right)\right]$$

et le vecteur $\mathbf{b}(v_r)$ est défini par

$$\mathbf{b}(v_r) = \left[1 \ \exp\left(\frac{j2\pi 2v_r T_r}{\lambda}\right) \ \ldots \ \exp\left(\frac{j2\pi(M-1)2v_r T_r}{\lambda}\right)\right]$$

Le terme $\psi = (D/\lambda)\sin\theta$ représente la fréquence spatiale du signal alors que le terme $\omega = (2v_r/\lambda)T_r$ représente la fréquence temporelle (Doppler) du signal.

Si α est l'amplitude complexe de la cible et $\mathbf{n}$ le vecteur des échantillons de bruit thermique (voir Annexe A.1.2), le vecteur reçu $\mathbf{x}$, vecteur des échantillons marqués d'un point rouge sur la Figure 2.1, s'écrit

$$\mathbf{x} = \alpha \mathbf{s_s}(\theta, v_r) + \mathbf{n} \tag{2.2}$$

Dans ces conditions, le traitement standard (formation de voies et traitement Doppler) consiste à effectuer l'opération suivante

$$y(\theta, v_r) = \mathbf{s_s^H}(\theta, v_r)\mathbf{x} \tag{2.3}$$

Nous comparons ensuite la quantité $|y(\theta, v_r)|^2$ à un seuil de détection η en fonction du taux de fausse alarme désiré [9].

2.1.2 Cas avec interférence : présence de fouillis

Si le bruit n'est pas blanc, ce qui est le cas du fouillis comme nous l'avons vu au Chapitre 1, c'est à dire que le bruit n'est pas réparti de manière égale dans le plan (θ, v_r). Nous pouvons le caractériser par sa matrice de corrélation appelée également matrice de covariance, qui mesure la corrélation entre tous les couples d'échantillons spatio-temporels.

$$\mathbf{\Gamma} = E\{\mathbf{n}\mathbf{n^H}\} \tag{2.4}$$

Le filtre optimal STAP, réjecteur du fouillis (ou plus généralement des interférences), au sens de la variance minimale est donné par [10] et [11]

$$\mathbf{w_{opt}} = \kappa \mathbf{\Gamma^{-1}} \mathbf{s_s}(\theta, v_r) \tag{2.5}$$

où $\kappa = 1/(\sqrt{\mathbf{s_s^H}(\theta, v_r)\mathbf{\Gamma^{-1}}\mathbf{s_s}(\theta, v_r)})$ est un scalaire de normalisation. Cette normalisation au dénominateur est nécessaire car après blanchiment, les répliques blanchies $\mathbf{s_s}(\theta, v_r)$ n'ont plus nécessairement toutes la même énergie alors que nous souhaitons mesurer la ressemblance à la réplique et non l'énergie qui est fonction de la répartition des parasites dans le plan (θ, v_r). Ce traitement est appelé *Sample Matrix Inversion* (SMI) lorsque $\mathbf{\Gamma}$ est estimée à partir des données, et correspond à un traitement STAP pré-Doppler sur un bloc de données spatio-temporelles de taille NM.

Pour chaque couple (θ, v_r), le traitement STAP consiste donc à calculer

$$\mathbf{y}(\theta, v_r) = \frac{\mathbf{s_s^H}(\theta, v_r)\mathbf{\Gamma^{-1}x}}{\sqrt{\mathbf{s_s^H}(\theta, v_r)\mathbf{\Gamma^{-1}s_s}(\theta, v_r)}} \tag{2.6}$$

et à comparer $|y(\theta, v_r)|^2$ à un seuil de détection η qui dépend du taux de fausse alarme désiré.

Dans la pratique, la vraie matrice de covariance $\mathbf{\Gamma = \Gamma^H}$ n'est pas connue a priori. Si les interférences sont essentiellement constituées d'échos de fouillis, nous pouvons faire l'hypothèse que les statistiques des échos de fouillis varient peu d'une case distance à une autre. Nous pouvons alors estimer la matrice de covariance à partir des échantillons de données pris dans des cases distances voisines adjacentes à la case sous test. Ces données sont appelées données d'entraînement à la différence des données primaires qui constituent la case distance sous test. Il est évident que le choix des données d'entrainement est crucial pour les performances du filtre STAP.

Si $\mathbf{x_k}$ est le vecteur des signaux reçus à la case distance k et si nous utilisons K cases distances pour estimer la matrice de covariance $\mathbf{R}$ avec l'estimateur *Sample Covariance Matrix* (SCM), la matrice s'écrit

$$\mathbf{R} = \mathbf{R}_{SCM} = \frac{1}{K}\sum_{i=1}^{K} \mathbf{x}_i \mathbf{x}_i^H \tag{2.7}$$

La matrice estimée $\mathbf{R}$ remplace alors la matrice théorique $\mathbf{\Gamma}$ dans l'équation (2.6) et la sortie du filtre devient

$$y(\theta, v_r) = \frac{\mathbf{s_s^H}(\theta, v_r)\mathbf{R^{-1}x}}{\sqrt{\mathbf{s_s}(\theta, v_r)\mathbf{R^{-1}s_s^H}(\theta, v_r)}} \tag{2.8}$$

et nous comparons toujours $|y(\theta, v_r)|^2$ à un seuil de détection η qui dépend du taux de fausse alarme désiré. Ce détecteur est appelé *Adaptive Matched Filter* (AMF) [11].

2.2 Traitement STAP de référence et notations

Comme nous l'avons vu, le traitement STAP réalise un filtrage adaptatif bidimensionnel, où différentes voies spatiales (sous-réseaux de l'antenne) sont combinées à des instants différents (appelés nombre d'impulsions du filtre STAP ou nombre de retards STAP) par l'intermédiaire de coefficients de filtrage estimés de manière adaptative sur des données d'entrainement. Ces données sont choisies dans des cases distance adjacentes à la case distance traitée, appelée case sous test (CST). L'estimation de ces coefficients est toujours déduite, plus ou moins directement, de la matrice de covariance du signal reçu du fouillis, qui est la quantité clé dans le processus de filtrage adaptatif. Un traitement STAP doit toujours rester cohérent avec la stratégie du traitement radar visant à obtenir une probabilité de détection P_d élevée tout en maintenant une très faible probabilité de fausses alarmes P_{fa}.

Pour atteindre cet objectif, le traitement STAP retenu devra donc idéalement respecter les étapes suivantes : sélectionner des données d'entrainement dont les signaux parasites (fouillis, éventuellement brouilleurs) possèdent les mêmes statistiques que la CST, utiliser ces données d'entrainement pour construire un filtre qui supprime les signaux parasites tout en conservant le signal cible potentiel dans la CST, détecter des cibles en comparant le niveau de puissance de la CST après filtrage STAP.

2.2.1 Données et notations

Nous définissons maintenant les conventions et notations qui seront utilisées dans la suite :

- N représente le nombre de voies spatiales, i.e le nombre de sous-réseaux de l'antenne,
- M représente le nombre de voies temporelles choisies pour le filtre STAP, aussi appelé nombre d'impulsions ou nombre de retards,
- M_p représente le nombre d'impulsions total dans une rafale cohérente d'impulsions,
- L représente le nombre de cases distance total.

Classiquement, les données radar sont représentées sous forme d'un cube de données (Figure 2.2)

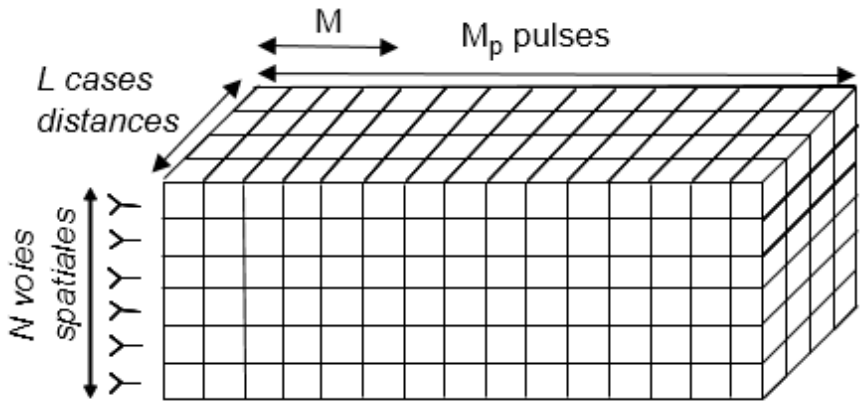


FIGURE 2.2 – Cube de données radar

Pour le traitement STAP, nous pouvons ré-arranger ce cube de données de manière à faire apparaitre des vecteurs de données spatio-temporelles de dimension NM (voir Figure 2.3). De cette façon, pour chaque case distance, la matrice de données $\mathbf{X}$ est de taille $NM \times K_t$, K_t étant le nombre d'échantillons utilisés pour estimer la matrice de covariance sur une case distance, le long des impulsions. Cela conduit au modèle de signal suivant avec les deux hypothèses H_0 sans cible et H_1 avec une cible présente

$$\begin{aligned} \mathbf{X} &= \mathbf{N}, && (H_0) \\ \mathbf{X} &= \alpha \mathbf{s_s} \mathbf{s_t^T} + \mathbf{N} && (H_1) \end{aligned} \tag{2.9}$$

$\mathbf{s_s}$ est le *steering* vecteur spatio-temporel (taille NM), $\mathbf{s_t}$ est le *steering* vecteur temporel (taille $K_t = M_p - M + 1$) et $\mathbf{N}$ est la matrice de données contenant les interférences (fouillis et bruit). La matrice de covariance estimée s'écrit alors $\mathbf{R} = (\mathbf{X}\mathbf{X}^\mathbf{H})/K_t$.

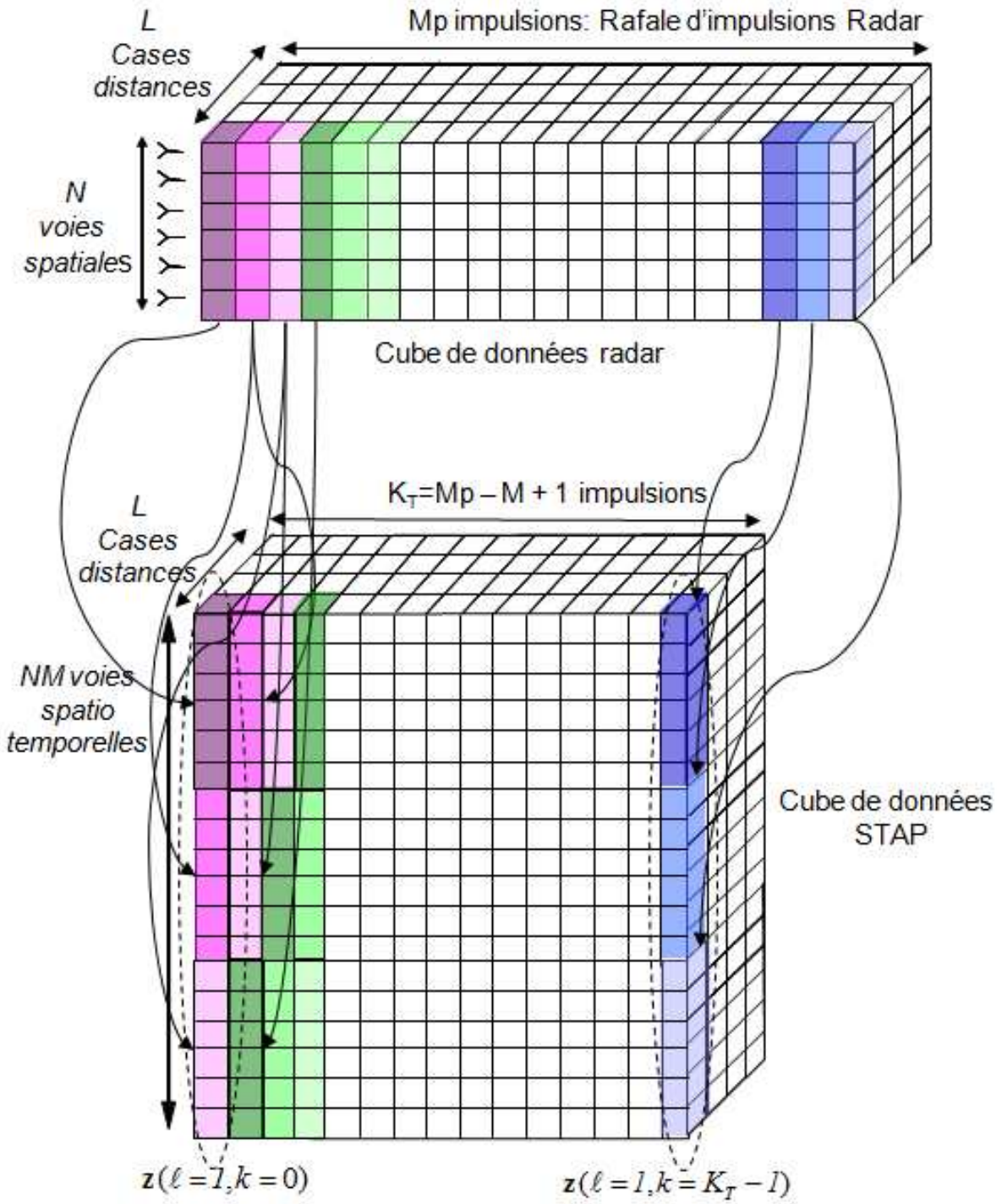


FIGURE 2.3 – Réorganisation des données selon le formalisme donné dans (2.9)

2.2.2 Traitement STAP de référence

Les traitements que nous proposerons seront systématiquement comparées aux performances d'un détecteur standard pris comme référence : les *Space-Time FIR Filters* décrits dans [12], utilisant comme données secondaires les cases distance voisines de la CST. Soit $\mathbf{R}$ une matrice inversible, le problème de minimisation du critère $\mathbf{w^H R w}$ qui représente la puissance en sortie des interférences et du bruit sous contrainte $\mathbf{C^H w = f}$ selon $\mathbf{w}$ donne comme résultat

$$\mathbf{w^H} \;=\; \mathbf{f^H (C^H R^{-1} C)^{-1} C^H R^{-1}}$$

La minimisation de la puissance des interférences en sortie sous contrainte de gain unité sur le signal $\mathbf{w^H s_s} = 1$ donne comme solution

$$\mathbf{w^H} = \frac{\mathbf{s_s^H R^{-1}}}{\mathbf{s_s^H R^{-1} s_s}} \tag{2.10}$$

qui une fois normalisée par la puissance du résidu en sortie donne le détecteur AMF vu dans (2.5) et (2.6)

$$\text{Détecteur AMF} \;:\; \frac{|\mathbf{s_s^H R^{-1} x}|^\mathbf{2}}{\mathbf{s_s^H R^{-1} s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{2.11}$$

Dans le cas du FIR, la solution pour la minimisation sous contrainte de diagramme au repos $\mathbf{w^H \Gamma_{th} w_q} = \mathbf{w_q^H \Gamma_{th} w_q}$ où $\mathbf{\Gamma_{th}}$ est la vraie matrice de covariance du bruit thermique (c.f Annexe A.1.2) et $\mathbf{w_q}$ est le vecteur des poids au repos (c'est à dire des poids dans le cas sans interférences où aucun filtrage n'est nécessaire) donne

$$\mathbf{w^H} = (\mathbf{w_q^H \Gamma_{th} w_q}) \frac{\mathbf{w_q^H \Gamma_{th} R^{-1}}}{\mathbf{w_q^H \Gamma_{th} R^{-1} \Gamma_{th} w_q}} \tag{2.12}$$

où $\mathbf{w_q}$ est défini par

$$\mathbf{w_q^H} = [\underbrace{11\ldots111}_{N\ fois}\underbrace{00000\ldots000000000}_{(M-1)N\ fois}]$$

Si les données d'entrainement ne contiennent que du bruit thermique, i.e $\mathbf{R} \simeq \mathbf{\Gamma_{th}}$, alors $\mathbf{w^H} \simeq \mathbf{w_q^H}$. Le détecteur associé aux filtres STAP FIR s'écrit alors comme précédemment par comparaison entre la puissance du signal blanchi et la puissance estimée du résidu de signal parasite au niveau du vecteur de pointage $\mathbf{w^H R w}$, appelée *Adaptive Power Residue* ou APR (voir Annexe B.5.2). Le test de détecteur est donc

$$\text{Détecteur FIR} \;:\; \frac{|\mathbf{w_q^H \Gamma_{th} R^{-1} x}|^2}{\mathbf{w_q^H \Gamma_{th} R^{-1} \Gamma_{th} w_q}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{2.13}$$

Ce détecteur FIR sera utilisé dans la suite comme traitement de référence STAP.

2.3 Problématique des environnements hétérogènes

Pour obtenir de bonnes performances, un traitement STAP doit respecter certaines contraintes. En particulier, les données d'entrainement doivent être statistiquement semblables, en nombre suffisant et ne pas contenir de signal utile. En pratique, il est souvent difficile de satisfaire pleinement ces contraintes à cause de l'hétérogénéité des environnements.

2.3.1 Environnement hétérogène : forte densité de cibles

Comme nous l'avons vu, les traitements STAP utilisent les données des cases distances voisines à la case sous test pour estimer les poids du filtre STAP. Les signaux issus des cibles mobiles ne devant pas être supprimés par le filtre STAP, ces signaux ne doivent pas être présents dans les données d'entrainement. Dans les zones à forte densité de cibles, la probabilité que des cibles de même direction et de même vitesse (fréquence Doppler) soient présentes dans des cases distance voisines est non négligeable. Dans ce cas, les données contenant du signal utile serviront à estimer la matrice de covariance $\mathbf{R}$ et le signal utile de la case sous test sera d'autant plus atténué qu'il y a de signal utile dans la matrice de covariance. Ces situations se rencontrent notamment avec les axes de communications (routes, autoroutes) ainsi qu'avec les convois. Les figures suivantes illustrent ce problème. Sur la Figure 2.4 sont affichées les positions des 10 cibles mobiles formant un convoi sur une image distance-vitesse non filtrée. Ces cibles ont une vitesse faible et sont donc noyées dans le fouillis. Elle ne sont pas détectables avec un traitement Doppler seul.

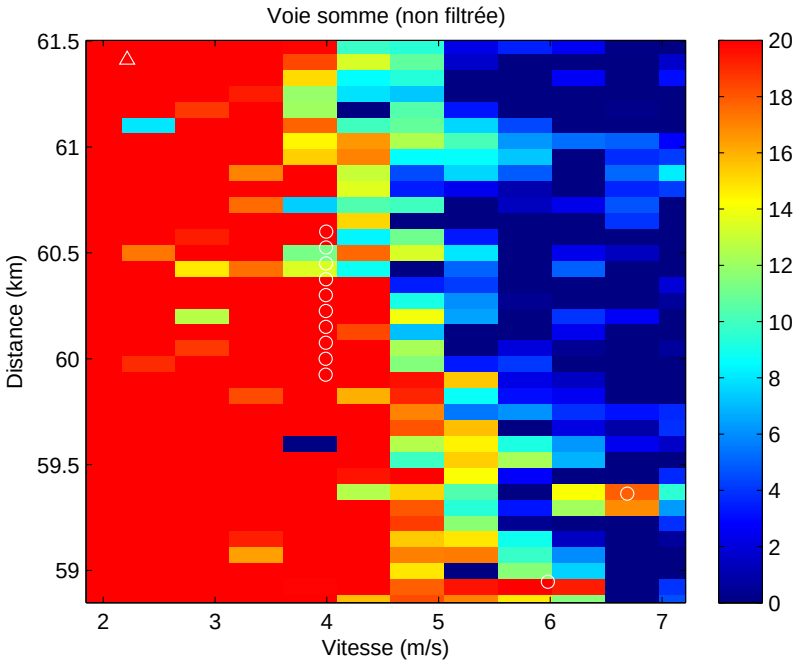


FIGURE 2.4 – Zoom d'une image distance-vitesse sans filtrage. Le convoi (série de cercles blancs) est noyé dans le fouillis.

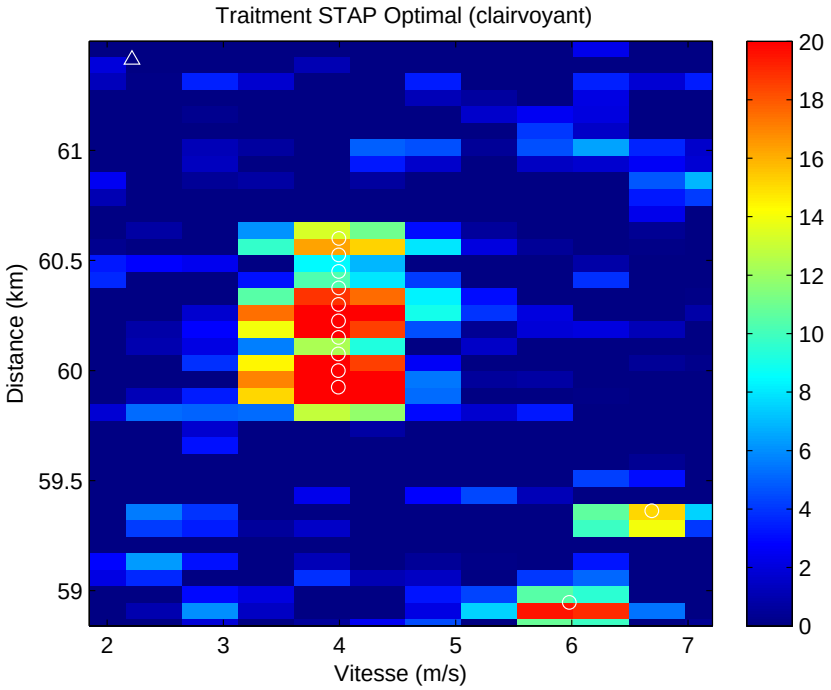


FIGURE 2.5 – Zoom d'une image distance-vitesse avec traitement STAP optimal.

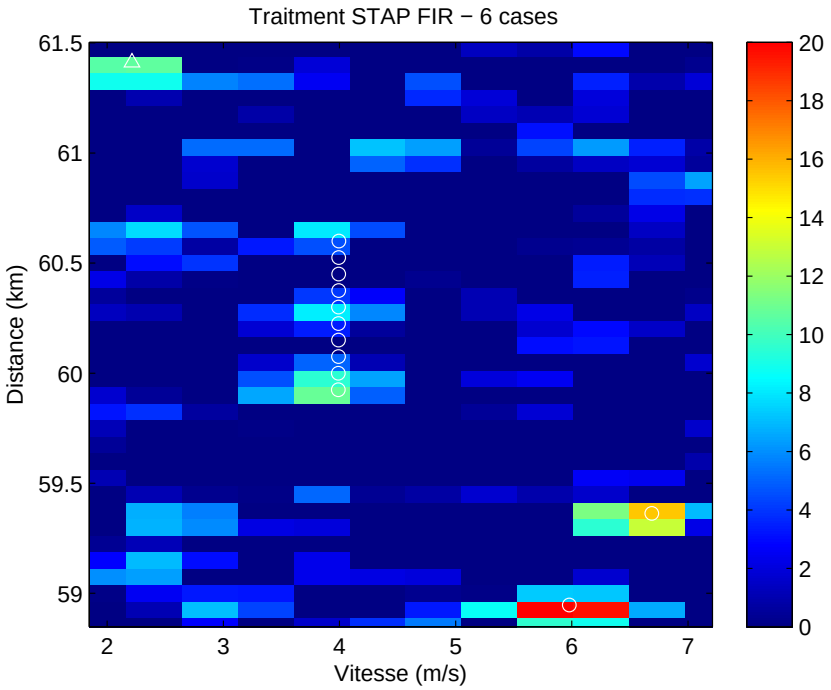


FIGURE 2.6 – Zoom d'une image distance-vitesse avec traitement STAP FIR utilisant 6 cases distance comme données d'entrainement.

Sur la Figure 2.5, le traitement STAP optimal, qui utilise la vraie matrice de covariance permet de supprimer le fouillis et de détecter les cibles du convoi. Il permet aussi de détecter les deux cibles se trouvant en bas à droite de l'image dont une qui était déjà détectable avec un traitement Doppler seul. La Figure 2.6 montre le résultat du traitement FIR utilisant 6 cases distance adjacentes (avec 1 case de garde de chaque coté) pour l'estimation de la matrice de covariance. Le fouillis est correctement supprimé, cependant, à cause de la présence de cibles dans ces cases adjacentes, le signal utile est presque totalement atténué et le convoi n'est plus détectable. En revanche, les deux cibles isolées se trouvant en bas à droite de l'image sont préservées et détectables.

2.3.2 Environnement hétérogène : fouillis non-stationnaire

L'obtention de données d'entraînement représentatives du fouillis de la case sous test est rendue complexe par des hétérogénéités intrinsèques du fouillis comme la présence inévitable d'échos ponctuels sur le fond diffus, la présence de transition de fouillis (urbain-rural, sol-mer, masques, reliefs, etc) et la présence d'un mouvement propre (arbres, vagues, etc). En plus de ces hétérogénéités intrinsèques, des hétérogénéités liées à la configuration du radar, comme par exemple l'utilisation d'une antenne frontale ou la présence d'ambiguïté distance, peuvent se rajouter. Quelle que soit l'origine de l'hétérogénéité, le résultat est une non-stationnarité en distance du fouillis. Il devient alors difficile de choisir des cases distance pour estimer la matrice de covariance. La Figure 2.7 montre une image distance-vitesse obtenue avec les données réelles **SAREX** où nous pouvons clairement observer la non-stationnarité du fouillis en distance.

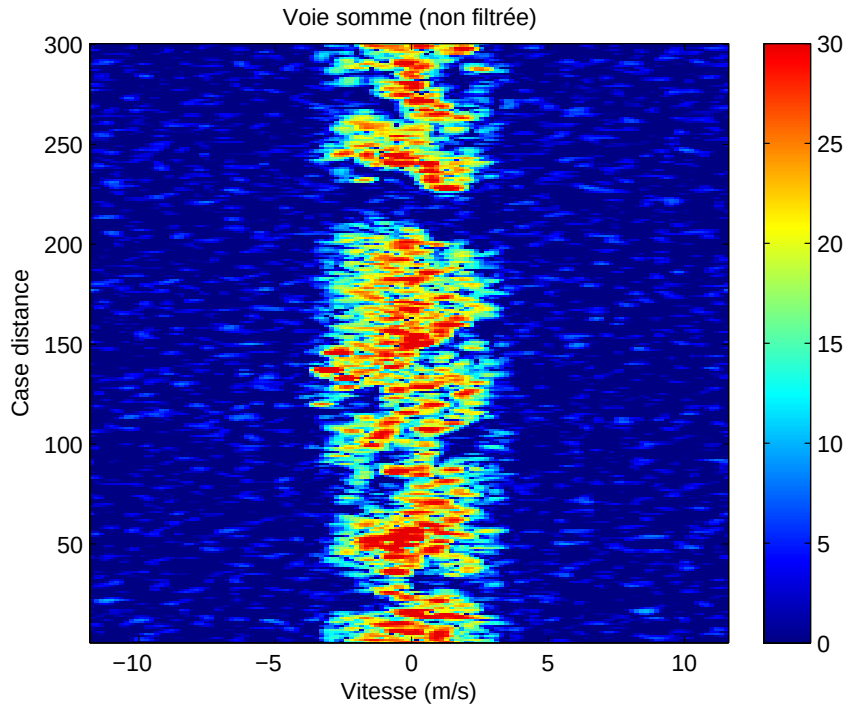


FIGURE 2.7 – Image distance-vitesse illustrant un fouillis non-stationnaire en distance.

2.4 Traitement STAP en environnement hétérogène : état de l'art

Nous présentons ici plusieurs méthodes visant à surmonter les problèmes liés à l'hétérogénéité de l'environnement. Une présentation plus exhaustive de telles méthodes est faite dans [13].

2.4.1 Méthodes à faible support de données d'entrainement

Pour lutter contre l'hétérogénéité du milieu, une première voie est d'utiliser des méthodes nécessitant peu de données d'entrainement. Parmi ces méthodes, la plus simple est la méthode du *diagonal loading* qui est présentée en Annexe B.4 où un facteur de chargement est ajouté à tous les termes diagonaux de la matrice de covariance. Cette méthode permet de diminuer le domaine d'estimation mais le choix du facteur de chargement reste un point dur [14].

Les algorithmes à rang réduit sont particulièrement bien adaptés au traitement STAP. Ils permettent de projeter les données dans un espace de dimension réduite. Nous pouvons passer d'un domaine d'estimation de taille $2NM$ à une taille $2p$ (avec p la dimension du sous-espace à atténuer) tout en maintenant une perte de 3dB sur le rapport signal à bruit par rapport au filtre optimal. Parmi les méthodes à rang réduit, nous pouvons citer : l'*eigencanceller* (EC) [15] basé sur la décomposition en éléments propres de la matrice de covariance, la méthode *Cross Spectral Metric* (CSM) [16] qui essaie de sélectionner les p vecteurs propres de la matrice de covariance qui maximisent le SINR, ou la méthode *Multistage Wiener Filter* (MWF) [17] qui consiste à trouver une base permettant de représenter le signal de la façon la plus compacte possible. Tous ces algorithmes requièrent cependant la connaissance du rang p de la matrice de covariance, ce point est crucial pour l'implémentation de tels algorithmes. La règle de Brennan [18] permet de donner une valeur approximative théorique de ce rang mais divers phénomènes peuvent mener à une augmentation de celui-ci.

De plus, ces méthodes permettent de réduire le nombre d'échantillons nécessaire à l'estimation de la matrice de covariance et donc de réduire l'impact négatif qu'ont les environnements hétérogènes sur les performances des traitements STAP mais sont elles aussi limitées dans les environnements fortement hétérogènes et ne résolvent pas la problématique des zones à forte densité de cibles.

2.4.2 Méthodes de sélection des données secondaires

En environnement hétérogène, la sélection des données d'entrainement peut permettre d'améliorer sensiblement les performances du traitement STAP. La méthode non-adaptative appelée *overnulling* [19] sélectionne les cases distance pour lesquelles l'équation radar (voir Annexe A.1.1) prédit les plus fortes puissances de fouillis.

Les méthodes de sélection adaptatives existent aussi comme la méthode *Power Selected Training* (PST) [20] ou la méthode de sélection de données d'entrainement présentée dans [21] qui consistent toutes les deux à calculer la puissance par case distance à par-

tir du signal reçu et de sélectionner les données d'entrainement à partir de ce critère. L'utilisation de la puissance comme critère principal de sélection peut néanmoins être limitant dans certaines situations pratiques où le fouillis est de même puissance mais réparti différemment dans le plan angle-Doppler.

Enfin, les détecteurs de non-homogénéité (NHD, *Non-Homogeneity Detector*) comme le *Generalizerd Inner Product* (GIP)[22] ou l'APR [23] sont basées sur des métriques plus complexes que les méthodes citées précédemment. Elles reposent sur l'identification des données hétérogènes par rapport à un jeu de données supposées homogènes entre elles. Ces dernières méthodes sont efficaces pour détecter les changements de milieu mais ne peuvent pallier à l'absence de données représentatives et ne peuvent rien faire contre la présence de cibles dans les cases distances voisines. De plus, elles requièrent une charge de calcul élevée parce qu'elles comparent les vecteurs de données à une matrice de covariance et ne comparent pas deux matrices de covariance entre elles, nous calculons ainsi une valeur moyenne pour chaque case distance.

2.4.3 Détecteurs basés sur un modèle d'hétérogénéité

Une autre stratégie consiste à prendre en compte le modèle de l'hétérogénéité dans la construction du détecteur. Deux modèles qui permettent de prendre en compte la variation du signal en fonction de la distance existent dans la littérature.

Le premier type de modèle considère qu'il existe un facteur constant et inconnu entre la matrice de covariance des données primaire et la matrice de covariance des données secondaires (des cases distance voisines). Le détecteur *Adaptive Coherent Estimator* (ACE) est alors le détecteur GLRT de ce modèle et est défini dans [24] par

$$T_{ACE}(x) = \frac{|\mathbf{s_s^H R^{-1} x}|^2}{(\mathbf{s_s^H R^{-1} s_s})(\mathbf{x^H R^{-1} x})} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta_{ACE} \tag{2.14}$$

Le test ACE correspond à un produit scalaire (mesure d'angle) entre la donnée primaire et la cible sous test après blanchiment par la matrice de covariance $\mathbf{R}$. Ce détecteur possède la propriété de taux de fausse alarme constant (CFAR) et est invariant par rapport au changement de puissance de la case sous test ou des données secondaires. Il est aussi le détecteur GLRT (voir Annexe A.1.3) pour d'autres types de modèle d'environnement [25][26].

Le second type de modèle est basé sur un modèle de texture défini comme un processus gaussien-composé dont la texture est aléatoire. Ce modèle, appelé SIRV pour *Spherically Invariant Random Vectors*, permet de prendre en compte des variations plus rapides du fouillis. Le modèle pour le vecteur gaussien-composé $\mathbf{c}$ est le suivant

$$\mathbf{c} = \sqrt{\tau}\mathbf{x}$$

où τ et $\mathbf{x}$ sont indépendants entre eux et où la distribution de τ n'est pas forcément connue. Ce modèle est plus adapté aux variations de puissances du fouillis impulsionnelles. Des nouveaux estimateurs ont été proposés à partir de ce modèle.

En particulier l'estimateur du point fixe [27] qui calcul la matrice de covariance de manière récursive s'écrit avec m la taille du vecteur $\mathbf{x}_i$

$$\mathbf{R}_{FP} = \frac{m}{K} \sum_{i=1}^{K} \frac{\mathbf{x}_i \mathbf{x}_i^H}{\mathbf{x}_i^H \mathbf{R}_{FP}^{-1} \mathbf{x}_i} \tag{2.15}$$

Dans [28], les auteurs montrent que ces estimateurs donnent de meilleures performances que l'estimateur SCM (voir équation (2.7)) dès lors que les données secondaires contiennent des perturbations ou qu'elles sont non gaussiennes.

2.5 Conclusion

Les différentes méthodes que nous avons présentées permettent de réduire l'impact négatif qu'ont les environnements hétérogènes sur le traitement STAP. L'utilisation de certains algorithmes vus dans les Sections 2.4.1 et 2.4.2 permettent de réduire le nombre de données d'entrainement nécessaires et de mieux les sélectionner. Les méthodes basées sur des modèles de fouillis permettent quant à elles d'améliorer les performances dans une certaine mesure. Cependant, ces modèles font l'hypothèse que les signaux reçus sur les cases distance adjacentes sont indépendants et distribués de manière identique au fouillis de la case sous test, éventuellement à une variable de texture près dans le cas des SIRV. En particulier, les modèles SIRV les plus élaborés, qui introduisent une variable de texture, sont mal adaptés à la représentation du fouillis en présence d'ambiguïtés distance (la composante gaussienne du fouillis est elle même rapidement variable en distance) et à la présence d'un très grand nombre de cibles.

Nous considérerons, dans notre approche, que l'environnement est trop hétérogène pour utiliser des données de cases distance adjacentes comme données d'entrainement. Nous développerons pour cela plusieurs méthodes n'utilisant que les données primaires. Nous étudierons ensuite la possibilité d'utiliser des données secondaires lorsque cela est possible. Cette approche n'est, comme nous allons le voir, pas incompatible avec la plupart des méthodes citées précédemment.

Chapitre 3

Traitement temps-réel : généralités et état de l'art

Sommaire

L'implémentation matérielle des traitements radar est un point crucial en aéroporté en raison du volume restreint disponible combiné à une forte demande en puissance de calcul. L'augmentation de la complexité des systèmes radar, tout comme le traitement du signal associé constituent les obstacles à l'utilisation de traitements STAP. Dans ce chapitre, nous définissons le contexte des traitements radar embarqués, puis nous dressons un état de l'art des implémentations de traitement radar sur GPU.

3.1 Contexte temps-réel embarqué

3.1.1 Introduction

Les traitements embarqués, comme le traitement radar, demandent de plus en plus de puissance de calcul sous des contraintes d'encombrement, de poids et de consommation (*Size, Weight and Power* ou SWaP) toujours plus faibles. Les circuits logiques programmables FPGA et les processeurs graphiques GPU sont de plus en plus utilisés pour le traitement parallèle et répétitif des données. Les FPGAs ont la capacité d'intégrer de la logique, des fonctions DSP ainsi que du calcul généraliste ce qui réduit la consommation et donc le SWaP. Les processeurs graphiques ont eux été utilisés depuis longtemps pour l'affichage vidéo des applications militaires. Depuis que ces processeurs ont la capacité de faire du calcul générique, les GPUs sont exploités pour du traitement d'image ou de signal qui était généralement le domaine des FPGAs. Alors que les FPGAs et GPUs ont chacun des avantages, chacun apporte une problématique technique et économique pour un utilisation pratique.

3.1.2 FPGA et GPU : avantages et inconvénients

Plusieurs familles de FPGA sont disponibles couvrant un grand éventail de performances et de coûts, permettant aux concepteurs de choisir exactement le FPGA qui correspond à leur application en fonction des ressources nécessaires, telles que les entrées-sorties ou les unités logiques. Une seule famille de FPGA offre une gamme allant de petites matrices, et donc moins consommatrices et moins chères, à de grandes matrices, permettant au concepteur de choisir dans une famille la taille et donc la performance et le coût de son système de calcul.

Les processeurs GPU quant à eux, tendent à être généralement de plus grande taille, les rendant moins flexibles et il est donc plus difficile à faire correspondre exactement leur taille à une application donnée. L'utilisation des GPUs pour le calcul générique étant très majoritairement faite pour la simulation ou la charge de calcul est très grande, les GPUs ont pour le moment plutôt concurrencé les super-ordinateurs dotés d'un grand nombre de processeurs généralistes CPU. Ce sont donc les modèles de GPU les plus puissants qui ont été implémentés dans les cartes orientées calcul, laissant les GPUs plus petits, d'entrée ou de milieu de gamme, beaucoup moins utilisés pour le calcul générique. Les GPUs, riches de fonctionnalités intégrées, sont beaucoup plus faciles à programmer que les FPGAs. Les GPUs du fabricant NVIDIA utilisent un langage de programmation appelé CUDA qui pointe une des principales faiblesses des systèmes de traitement parallèles FPGA : la complexité de leur programmation. Le langage de programmation CUDA (voir Partie 3, Chapitre 2.3) permet aux programmeurs d'écrire les programmes avec des langages conventionnels et d'accéder aux capacités de traitement du GPU grâce à des extensions simples. La programmation est ainsi un peu plus complexe que sur un processeur généraliste CPU, mais bien plus facile que pour un FPGA. En effet, l'optimisation d'un FPGA nécessite l'un des deux langages de description de matériel, VHDL ou Verilog.

	AMD Phen. II 1090T CPU	AMD Radeon E6760 GPU	NVIDIA GeForce GT240 GPU (GRA 111)	Xilinx Virtex-6 SW475T FPGA
SP GFLOP/s	153.6	576	250	550
GFLOP/s / W	1.23	16.5	7.1	13.7
GFLOP/s / $	0.54	N/A	2.7	0.14
Langage	C/C++	OpenCL	CUDA	VHDL
Difficulté de programmation	Basique	Moyenne	Facile	Difficile
Communauté de programmeurs	Très importante	Moyenne	Importante	Spécialisée
Coûts outils de programmation	Faible	Faible	Faible	Elevé
Débit Entrés/sorties	Limité par la mémoire RAM	Limité par le bus PCIex	Limité par le bus PCIex	Elevé
Pérennité des composants et des outils	Longue (versions serveurs)	N/A	N/A	Longue

FIGURE 3.1 – Comparaison de certaines caractéristiques d'un CPU AMD Phenom 1090T, d'un GPU AMD Radeon E6760, d'un GPU NVIDIA GeForce GT240 (équipant la carte de calcul embarquée GE-IP GRA111 et d'un FPGA Xilinx Virtex-6 SW475T.

Les deux sont complexes et nécessitent l'expertise spécialisée de concepteurs en logique numérique, qui ajoute un coût et un temps de conception en raison du faible nombre de spécialistes. De plus, sur les FPGAs, toutes les entrés-sorties et l'infrastructure mémoire doivent être gérées avec ces langages de programmation complexe et la correction des erreurs de programmation peut se révéler aussi fastidieuse avec ce type de langage. Les outils de programmation FPGA sont aussi plus chers, en moyenne quelques dizaines de milliers d'euros par suite logicielle. Au minium, une suite entière d'outils, incluant un compilateur, un logiciel de simulation et de vérification, sont nécessaires avec en plus d'autres outils spécifiques qui dépendant du besoin. Avec les GPUs, seul un ordinateur équipé d'un GPU et d'un compilateur C est requis.

En revanche, les entrés-sorties des FPGAs peuvent être programmées et redéfinies pour s'adapter au besoin du programmeur. Les processeur GPUs n'utilisent quant à eux qu'un type de d'entrés-sorties : le port PCI Express (PCIe), parce qu'ils ont été conçus à l'origine pour le rendu graphique sur PC. Quelque soit le type d'entrés-sorties du système, le port PCIe doit être utilisé pour les communications avec le GPU et toutes les données doivent circuler par celui-ci.

Traditionnellement utilisés pour le rendu 3D des jeux vidéos, les GPUs sont des processeurs à la pointe en terme de performance et d'optimisation du SWaP. Bien que consommant plus qu'un processeur CPU, ils permettent souvent de remplacer plusieurs cartes de calcul à base de CPU, en délivrant un gain important de performances. Pour des algorithmes de traitement du signal traditionnels, comme les transformés de Fourier discrètes (FFT), les GPUs offrent des performances inégalées, et tout particulièrement en terme de performance par Watt. Dans les systèmes embarqués, une diminution du SWaP permet d'augmenter les performances des algorithmes, tout en augmentant l'autonomie, le rayon d'action et la charge utile du porteur. Les GPUs excellent dans les opérations mathématiques comme le traitement d'image sur des flux de données parallèles (ils sont conçus pour cela à l'origine) où sont exécutées des opération en virgule flottante répétitives. Sur les FPGAs, les opérations mathématiques sont implémentées via un calcul à virgule fixe, ce qui résulte en une différence significative en terme de précision ce qui peut limiter la dynamique des nombres utilisés et donc les applications possibles. Dans ce domaine, les unités de calcul à virgule flottante (simple et double précision) sont un grand avantage pour le GPU. Le tableau de la Figure, compare plusieurs aspects d'un CPU, de deux GPUs et d'un FPGA de même génération [29].

3.1.3 Conclusion

En conclusion, les GPUs et les FPGAs ont chacun des avantages et sont complémentaires. Aucun des deux n'offre seul une solution optimale pour couvrir toute la chaine d'opérations d'un système embarqué. Les FPGAs apparaissent comme intéressants pour les opérations proches des capteurs, comme la numérisation ou la décimation des signaux alors que les GPUs apparaissent plus intéressants pour les opérations mathématiques plus complexes comme traitement du signal et notamment le traitement STAP. Dans la Section suivante, nous dressons un état de l'art des implémentations GPU de traitements STAP.

3.2 STAP sur GPU : état de l'art

Les GPUs sont très performants pour effectuer des tâches simples sur un très grand nombre de données (voir Partie 3, Chapitre 2). Ainsi, les GPUs sont extrêmement performants pour des traitement comme la cross-corrélation [30], le traitement Doppler [31], ou le filtrage adaptatif purement spatial [32], sur de grands réseaux d'antennes fonctionnant à large bande. Dans ces applications, les GPUs apportent un gain important en rapidité d'exécution, la plupart du temps d'un facteur supérieur à 10.

Pour des applications de traitement radar aéroporté, les GPUs présentent un grand intérêt pour le traitement SAR [33][34]. Pour ces traitements, le GPU est particulièrement à l'aise dans le traitement de gros volumes de données. Dans [35], les auteurs montrent qu'un processeur GPU est 11 fois plus rapide qu'un processeur généraliste CPU de même gamme pour effectuer un traitement SAR.

En revanche, les traitements STAP ne sont pas nativement adaptés à l'architecture des GPUs. Les volumes de données sont faibles (voir Section 2, Figure 2.2) et il est difficile d'exploiter pleinement la puissance avec une parallélisation à bas niveau, comme elle doit être faite sur GPU. Dans [36], les auteurs évaluent les performances sur GPU de deux algorithmes couramment utilisés en STAP : la décomposition QR et la décomposition en valeurs singulières (voir Partie 2, Chapitre 2.2.2). Ces algorithmes sont utilisés sur les matrices de covariance pour l'inversion ou l'utilisation de méthodes à rang réduit. Les auteurs utilisent les fonctions de la bibliothèque CULA [37], pour lesquelles la parallélisation est faite à l'intérieur même des fonctions (bas niveau), et les matrices sont traitées à la suite. Les résultats montrent que le GPU apporte un gain à partir d'une taille de matrice de 200×200 et que ce gain devient important à partir d'une taille de 1000×1000. Ces résultats sont confirmés dans [38].

Nous avons mesuré les performances de la fonction *zheev* de la librairie CULA qui calcule la décomposition en éléments propres (EVD) d'une matrice de covariance hermitienne complexe.

La Figure 3.2 montre les temps de traitement pour une décomposition en éléments propres d'une matrice de covariance en fonction de sa taille. La calcul sur GPU est fait avec la fonction *zheev* de la librairie CULA, alors que sur CPU elle est faite avec la librairie mathématique Intel MKL. Les résultats nous font arriver aux mêmes conclusions que les auteurs de [36] : les GPUs sont performants sur des matrices de taille très importantes, ce qui n'est pas le cas des matrices de covariance en STAP, qui ont généralement une dimension inférieure à 100×100.

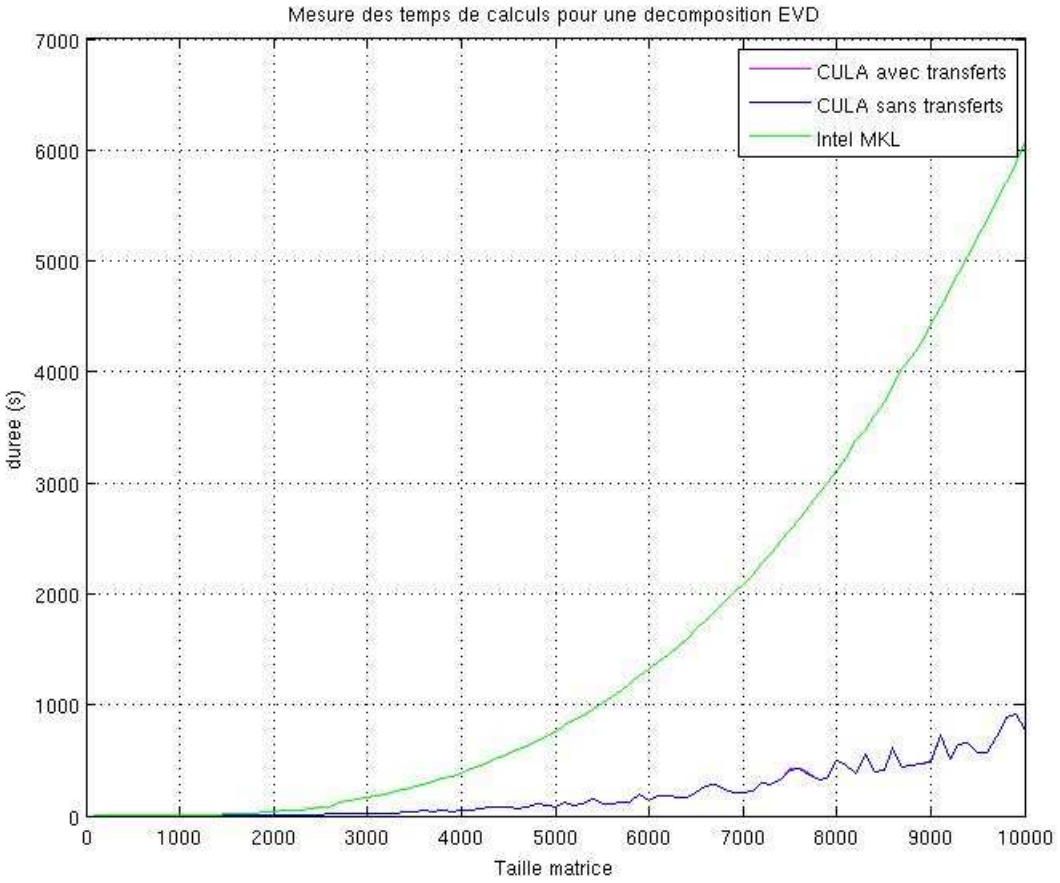


FIGURE 3.2 – Temps de traitement pour une décomposition en éléments propres d'une matrice de covariance en fonction de sa taille n (matrices de taille $n \times n$).

3.3 Conclusion

Nous avons vu que les processeur graphiques GPUs sont particulièrement intéressants pour effecteur du calcul sur des plateformes embarquées en raison de leur efficacité. En effet, ils proposent des puissances de calcul élevés pour un coût, un encombrement et une consommation réduite.

Les gains de performances que peuvent apporter les GPUs sur des applications de traitement du signal radar ont été démontrés pour de nombreuses applications de systèmes basés au sol, ou dans le cas du traitement SAR, de systèmes embarqués.

Les méthodes de parallélisation traditionnelles sur GPUs, liées à l'architecture de ces processeurs, ne les rendent a priori pas intéressants pour l'implémentation de traitements STAP. Nous développerons des implémentations des algorithmes STAP, en utilisant deux niveaux de parallélisation, afin d'exploiter la puissance des GPUs pour le calcul de petites matrices.

Deuxième partie

Traitement STAP en environnement hétérogène

1

Stop-Band APES : traitement pour environnent hétérogène

Sommaire

Dans un contexte de radar aéroporté, les situations hétérogènes sont un problème majeur à l'utilisation de traitements STAP pour lesquels les données secondaires requises ne doivent pas contenir de cibles et être homogènes aux données primaires. En conséquence, les performances des détecteurs STAP sont sévèrement impactées par les environnements hétérogènes. Dans ce chapitre, nous proposons une version étendue du *Maximum Likelihood Estimation Detector* (MLED) qui est particulièrement bien adapté à cette problématique puisqu'il travaille uniquement sur les données primaires. Par ailleurs, nous proposons une nouvelle méthode, appelée Stop-Band APES qui, permet de réduire la forte charge de calcul due à la haute résolution Doppler de la méthode MLED. Les performances de cette nouvelle méthode sont illustrées sur des données synthétiques réalistes.

1.1 Le détecteur MLED

La première étape pour la construction du détecteur MLED [39] consiste à mettre en œuvre l'algorithme *Amplitude and Phase Estimation* (APES) [40] d'estimation de l'amplitude α de la cible, obtenue par la recherche du filtre minimisant l'écart entre le signal temporel résiduel après filtrage et sa réplique temporelle, sous contrainte de préserver spatialement le signal d'intérêt. Mathématiquement, ce problème se formalise de la façon suivante :

$$\min_{\mathbf{w},\alpha}(\mathbf{w^H X} - \alpha \mathbf{s^T_{t(D)}})(\mathbf{w^H X} - \alpha \mathbf{s^T_{t(D)}})^H \text{ tel que } \mathbf{w^H s_s} = 1 \tag{1.1}$$

La solution obtenue est (voir Annexe B.1) :

$$\mathbf{w^H} = \frac{\mathbf{s_s^H Q^{-1}}}{\mathbf{s_s^H Q^{-1} s_s}} \text{ et } \alpha = \frac{\mathbf{w^H X s^*_{t(D)}}}{K_t} \tag{1.2}$$

où

$$\mathbf{Q} = \mathbf{R} - \mathbf{g g^H} \text{ et } \mathbf{g} = \frac{\mathbf{X s^*_{t(D)}}}{K_t} \tag{1.3}$$

$\mathbf{s_{t(D)}}$ est le *steering* vecteur de la case Doppler testée, α est une estimée de l'amplitude de la cible, par la méthode APES, $\mathbf{g}$ peut être obtenu simplement pour toutes les hypothèses de fréquence Doppler f_d par la transformée de Fourier sur l'axe temporel des données (transformée de Fourier suivant les lignes de $\mathbf{X}$), $\mathbf{Q}$ est une estimée de la matrice de covariance du signal interférence seul et $\mathbf{R}$ est une estimée de la matrice de covariance des données (signal utile et signal interférence).
Il est possible d'obtenir $\mathbf{Q^{-1}}$ à partir de $\mathbf{R^{-1}}$ en utilisant le lemme d'inversion matricielle (voir Annexe B.2) :

$$\mathbf{Q^{-1}} = \mathbf{R^{-1}} + \frac{\mathbf{R^{-1} g g^H R^{-1}}}{\mathbf{1 - g^H R^{-1} g}} \tag{1.4}$$

Le détecteur MLED s'écrit alors comme le rapport entre la puissance estimée de la cible, i.e. $|\alpha|^2$, et la puissance estimée du résidu de signal parasite au niveau du vecteur de pointage, i.e. $\mathbf{w^H Q w} = 1/(\mathbf{s_s^H Q^{-1} s_s})$.

Le test de détection se résume finalement à :

$$\text{Détecteur MLED} \ : \ \frac{|\mathbf{s_s^H Q^{-1} g}|^\mathbf{2}}{\mathbf{s_s^H Q^{-1} s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \ \eta \tag{1.5}$$

Il a été démontré [39][41] que ce test possédait la propriété requise de détecteur à taux de fausse alarme constant, et qu'il correspondait de plus au détecteur GLRT en l'absence de données secondaires pour un signal parasite interférant aléatoire Gaussien à moyenne nulle dans la case sous test.

1.2 Interprétation comme une minimisation de l'énergie dans l'espace orthogonal au sous-espace signal

L'insertion de l'estimée α, (1.2), de l'amplitude du signal utile, dans (1.1) amène à réécrire (1.1) sous la forme suivante :

$$\min_{\mathbf{w}} \mathbf{w^H} \left(\mathbf{X} - \mathbf{X} \frac{\mathbf{s^*_{t(D)} s^T_{t(D)}}}{\mathbf{s^T_{t(D)} s^*_{t(D)}}} \right) \left(\mathbf{X} - \mathbf{X} \frac{\mathbf{s^*_{t(D)} s^T_{t(D)}}}{\mathbf{s^T_{t(D)} s^*_{t(D)}}} \right)^H \mathbf{w} \text{ tel que } \mathbf{w^H s_s} = 1 \tag{1.6}$$

$$\iff \min_{\mathbf{w}} \left\{ \mathbf{w^H X} (\mathbf{I} - \mathbf{P}_{//}) (\mathbf{I} - \mathbf{P}_{//})^H \mathbf{X^H w} \right\} \text{ s.t } \mathbf{w^H s_s} = 1$$

$$\iff \min_{\mathbf{w}} \left\{ \mathbf{w^H X P_\perp P_\perp}^H \mathbf{X^H w} \right\} \text{ s.t } \mathbf{w^H s_s} = 1$$

où $\mathbf{s_{t(D)}}$ est le *steering* vecteur temporel de la case Doppler sous test et $\mathbf{P}_{//}$ est le projecteur dans l'espace signal de la case Doppler sous test :

$$\mathbf{P}_{//} = \frac{\mathbf{s^*_{t(D)} s^T_{t(D)}}}{\mathbf{s^T_{t(D)} s^*_{t(D)}}} = \frac{\mathbf{s^*_{t(D)} s^T_{t(D)}}}{K_t}$$

et $\mathbf{P}_\perp$ le projecteur dans l'espace orthogonal au signal :

$$\mathbf{P}_\perp = \mathbf{I} - \mathbf{P}_{//} = \mathbf{I} - \frac{\mathbf{s^*_{t(D)} s^T_{t(D)}}}{K_t}$$

La matrice estimée $\mathbf{Q}$ de l'équation (1.3) prend alors la forme suivante

$$\mathbf{Q} = \frac{\mathbf{X X^H}}{K_t} - \frac{1}{K_t} \mathbf{X P}_{//} \mathbf{X^H} \tag{1.7}$$

La nouvelle écriture de (1.6) donne un autre éclairage sur le détecteur MLED, qui apparaît comme une minimisation de l'énergie en sortie de traitement après projection des données dans l'espace orthogonal à l'espace signal, avec pour contrainte de conserver le signal d'intérêt. C'est cette projection qui supprime la composante signal utile pour l'estimation de la matrice de covariance $\mathbf{Q}$ du signal parasite seul, l'écriture de la matrice $\mathbf{Q}$ en (1.7) est la traduction implicite de cette projection. La finesse de cette suppression du signal utile se caractérise par la réponse en fréquence de la sortie du projecteur $\mathbf{P}_\perp$ lorsque l'entrée est constituée du signal utile $\mathbf{X} = \mathbf{s^T_t}(f_1)$. Cette réponse est en fait la transformée de Fourier en f de la sortie $\mathbf{Y} = \mathbf{X P}_\perp$ et on la note :

$$\tilde{\mathbf{P}}(f) = \mathbf{Y} \frac{\mathbf{s^*_t}}{\mathbf{s^T_t s^*_t}}$$

avec

$$\mathbf{Y} = \mathbf{s^T_t}(f_1) \left(\mathbf{I} - \frac{\mathbf{s^*_{t(D)} s^T_{t(D)}}}{K_t} \right)$$

En notant $\mathbf{s}^{\mathbf{T}}_{\mathbf{t(D)}} = \mathbf{s}^{\mathbf{T}}_{\mathbf{t}}(f_0)$, on a :

$$\tilde{\mathbf{P}}(f) = \mathbf{s}^{\mathbf{T}}_{\mathbf{t}}(f_1)\left(\mathbf{I} - \mathbf{s}^{*}_{\mathbf{t}}(f_0)\mathbf{s}^{\mathbf{T}}_{\mathbf{t}}(f_0)\right)\frac{\mathbf{s}^{*}_{\mathbf{t}}(f)}{K_t} \tag{1.8}$$

Cette réponse fréquentielle dépend des fréquences f_1 contenues dans le signal $\mathbf{X}$ et f_D correspondant à la case Doppler sous test.
Si $f = f_1$, nous évaluons la réponse à la case de la cible, et nous trouvons (voir Annexe B.3) :

$$\tilde{\mathbf{P}}(f_1) = 1 - \frac{1}{K_t}\frac{sin^2(\pi K_t(f_1 - f_0))}{sin^2(\pi(f_1 - f_0))} \tag{1.9}$$

L'allure de la réponse du projecteur est donnée finalement sur la Figure 1.1.

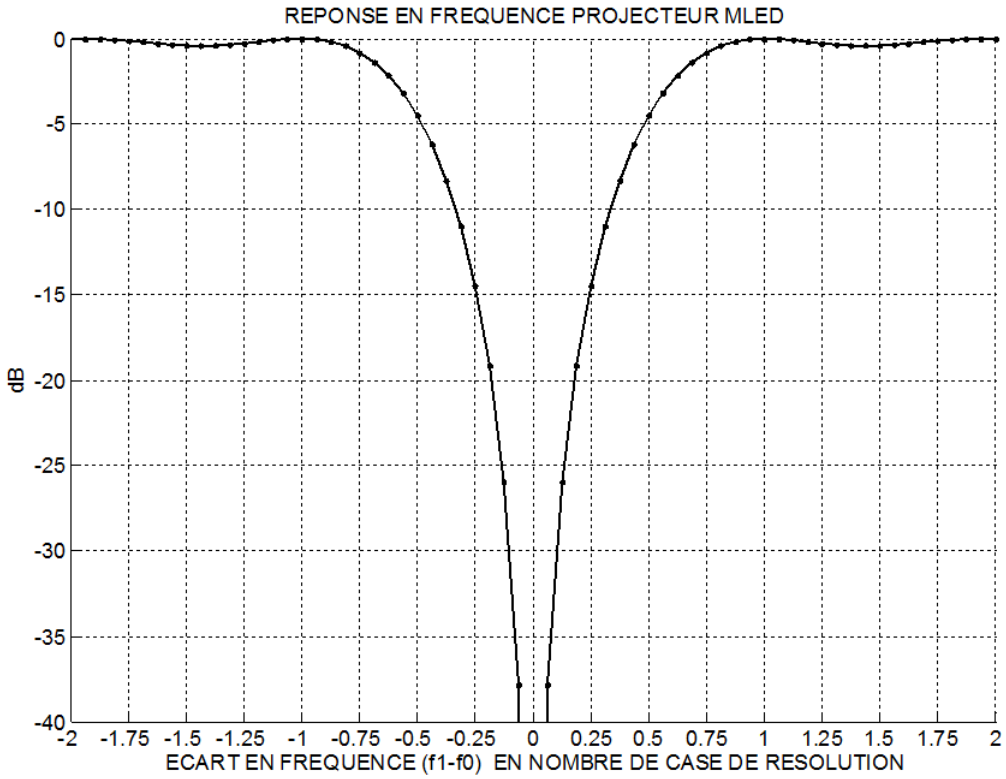


FIGURE 1.1 – Réponse en fréquence du projecteur impliqué dans le MLED

Une case de résolution représente une étendue de $1/K_t$ en fréquence Doppler normalisée. Cette figure met en évidence une très grande finesse du projecteur. Cette grande finesse est à l'origine des propriétés d'hyper-résolution de la méthode APES qui va imposer un test d'hypothèse avec une grille de pas bien plus fin que la case de résolution en fréquence afin de garantir l'élimination du signal cible dans l'estimation de la matrice signal parasite seul $\mathbf{Q}$.

Pour un pas d'une demi-case de résolution Doppler (sur-échantillonnage par 2 de l'analyse Doppler), la Figure 1.1 montre une atténuation de 15dB seulement d'une cible située au point milieu entre deux hypothèses de fréquence testées.
Pour un sur-échantillonnage par 4 de l'analyse Doppler, l'atténuation devient presque acceptable avec -25dB environ, et il faut un sur-échantillonnage de 8 pour atteindre des atténuations tout à fait satisfaisantes du signal cible de l'ordre de -40 dB.
On conclut des analyses précédentes, que la mise en œuvre du détecteur MLED nécessite au minimum un sur-échantillonnage Doppler d'un facteur 4, ce qui quadruple le nombre d'opérations de filtres STAP à appliquer (un par hypothèse Doppler). Des extensions du détecteur MLED, permettant de limiter ce sur-échantillonnage Doppler et donc le volume de calcul associé, sont présentées dans la section suivante.

1.3 Extension du MLED à Stop-Band APES

Cette extension est assez naturelle au vu des considérations suivantes. Elle consiste à étendre le projecteur dans l'espace signal intervenant dans le MLED aux fréquences voisines. Pour cela, nous introduisons un projecteur étendu :

$$\mathbf{P}_{//} = \frac{\mathbf{S_t^* S_t^T}}{\mathbf{S_t^T S_t^*}} \tag{1.10}$$

avec

$$\mathbf{S_t} = \mathbf{S_t}(f_D) = [\mathbf{s_t}(f_D - \nu_P) \ \cdots \ \mathbf{s_t}(f_D - \nu_1) \ \mathbf{s_t}(f_D) \ \mathbf{s_t}(f_D + \nu_1) \ \cdots \ \mathbf{s_t}(f_D + \nu_P)] \tag{1.11}$$

où $\nu_1 \ldots \nu_P$ sont les fréquences qui définissent l'étendue du projecteur $\mathbf{P}_{//}$.

Cela revient à écrire le problème de minimisation de l'équation (1.1) sous la forme

$$\min_{w,\alpha} (\mathbf{w^H X} - \alpha^{\mathbf{T}} \mathbf{S_t^T})(\mathbf{w^H X} - \alpha^{\mathbf{T}} \mathbf{S_t^T})^H \text{ s.t } \mathbf{w^H s_s} = 1 \tag{1.12}$$

avec la modélisation du signal qui devient

$$\mathbf{X} = \mathbf{s_s} \alpha^{\mathbf{T}} \mathbf{S_t^T} + \mathbf{N} \tag{1.13}$$

α est maintenant un vecteur d'amplitude.

La minimisation de (1.12) par rapport à α conduit à :

$$\alpha^{\mathbf{T}} = \mathbf{w^H X S_t^* (S_t^T S_t^*)^{-1}} = \mathbf{w^H G B^{-1}} \tag{1.14}$$

avec

$$\mathbf{G} = [\mathbf{g}_{-P} \ \cdots \ \mathbf{g} \ \cdots \ \mathbf{g}_P], \quad \mathbf{g}_i = \frac{\mathbf{X s_t^*}(f_0 - \nu_i)}{K_T}, \quad \nu_0 = 0, \ \mathbf{g_0} = \mathbf{g}, \quad i = 1, \ldots, P$$

La matrice $\mathbf{B} = \frac{\mathbf{S_t^T S_t^*}}{K_t}$ est indépendante de la case sous test. La réinsertion de α dans (1.12) amène bien alors à la minimisation :

$$\min_{\mathbf{w}} \left\{ (\mathbf{w^H X}(\mathbf{Id} - \mathbf{S_t^*}(\mathbf{S_t^T S_t^*})^{-1} \mathbf{S_t^T}) \mathbf{X^H w} \right\} \text{ tel que } \mathbf{w^H s_s} = 1 \tag{1.15}$$

L'équation (1.15) fait bien intervenir la minimisation de l'énergie en sortie de traitement après projection des données dans un espace orthogonal à un espace signal étendu (bande de fréquence définie par $\mathbf{S_t}$), d'où le qualificatif de Stop-Band APES pour ce détecteur. La solution de (1.15) est encore donnée par :

$$\mathbf{w^H} = \frac{\mathbf{s_s^H Q^{-1}}}{\mathbf{s_s^H Q^{-1} s_s}}$$

mais avec cette fois :

$$\mathbf{Q} = \frac{\mathbf{X X^H}}{K_t} - \frac{1}{K_t}\mathbf{X S_t^*}(\mathbf{S_t^T S_t^*})^{-\mathbf{1}}\mathbf{S_t^T X^H} = \mathbf{R} - \frac{1}{K_t}\mathbf{X P_{//} X^H} \tag{1.16}$$

$$\Longleftrightarrow \mathbf{Q} = \mathbf{R} - \mathbf{G B^{-1} G^H} \tag{1.17}$$

En toute rigueur, la détection devrait s'écrire comme précédemment par une comparaison entre la puissance estimée du signal utile, i.e. $\alpha^{\mathbf{T}}\alpha^*$, et la puissance estimée du résidu de signal parasite. Cependant, le modèle d'un signal à plusieurs composantes, (1.13), n'a été introduit que pour pallier les propriétés d'hyper résolution du MLED standard. En réalité, le signal n'ayant qu'une composante, il est préférable de conserver le même test de détection qu'auparavant :

$$\text{Détecteur Stop-Band APES} : \frac{|\mathbf{s_s^H Q^{-1} g}|^{\mathbf{2}}}{\mathbf{s_s^H Q^{-1} s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{1.18}$$

Sur la Figure 1.2, on présente l'allure de la réponse en fréquence des projecteurs étendus $\mathbf{P_{//}} = \mathbf{S_t^*}(\mathbf{S_t^T S_t^*})^{-\mathbf{1}}\mathbf{S_t^T}$

avec :

$$\mathbf{S}_t = [\mathbf{s_t}(f_0 - \frac{1}{2K_t}), \mathbf{s_t}(f_0), \mathbf{s_t}(f_0 + \frac{1}{2K_t})]$$

pour le projecteur étendu aux deux demi-cases de résolution voisines,

et avec :

$$\mathbf{S_t} = \mathbf{S_t}(f_0) = [\mathbf{s_t}(f_0 - \frac{1}{K_t}) \;\; \mathbf{s_t}(f_0) \;\; \mathbf{s_t}(f_0 + \frac{1}{K_t})]$$

pour le projecteur étendu aux deux cases de résolution voisines.

La Figure 1.2 précédente montre qu'en exploitant seulement un projecteur construit à partir des deux cases de résolution voisines, les atténuations dans la case sous test ne sont pas suffisantes (de l'ordre de 20 dB). En revanche, des atténuations supérieures à 50 dB pour tout signal situé dans la même case de résolution fréquentielle que la case sous test sont obtenues avec l'exploitation des deux demi-cases voisines.

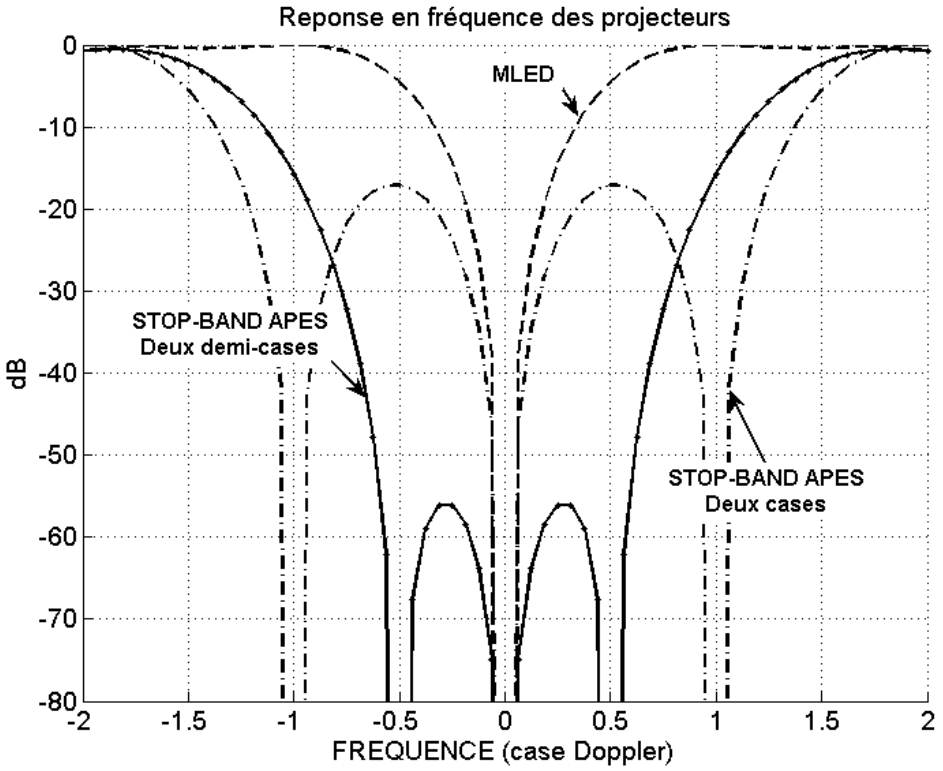


FIGURE 1.2 – Réponse en fréquence du projecteur impliqué dans le MLED

Nous pouvons donc conclure que l'exploitation de demi-cases de résolution est nécessaire. Cependant, par rapport au MLED, Stop-Band APES ne nécessitera pas de sur-échantillonnage par rapport à la résolution Doppler pour le calcul et l'application du filtre STAP. Un zéro-*padding* d'un facteur 2 sera néanmoins nécessaire au niveau des transformées de Fourier pour accéder aux signaux $\mathbf{g_i}$ qui doivent, eux, être évalués toutes les demi-cases de résolution pour la construction du projecteur.

Remarquons que, comme nous l'avons dit, ce projecteur supprime le signal d'intérêt de la matrice de covariance, c'est à dire qu'il supprime la tranche de signal qui se trouve à la case Doppler sous test. Plus le projecteur sera étendu, plus la tranche de signal retirée de la matrice de covariance sera importante. Le risque est que, si l'étendue du projecteur est trop large, une partie non-négligeable du signal parasite (fouillis) sera retiré de la matrice, entrainant ainsi une atténuation moindre de ce signal parasite par le filtre STAP. Ce point sera discuté et étudié dans le Chapitre 3.

1.4 Résultats

1.4.1 Résultats sur données DGA

Les différentes solutions algorithmiques présentées ont été mises en œuvre sur les données synthétiques DGA de l'essai 10 (voir Annexe A.3.2). Ces données tests ont été choisies pour le caractère illustratif (scénario simple et résultats facilement interprétables). Pour ces données, si les cibles sont synthétiques, le fouillis est quant à lui réel et obtenu à partir de données collectées en mode SAR dont on a dégradé les résolutions radiale et transverse. Sauf mention explicite, les figures de cette section correspondent toujours à un sur-échantillonnage d'un facteur 4 de l'analyse Doppler.

Sur la Figure 1.3, on peut observer la sortie du traitement Doppler classique de la voie somme habituelle. On peut noter la présence des pics au niveau des trois cibles, aux vitesses $-4\ m.s^{-1}$ et $+4\ m.s^{-1}$, ainsi que le spectre Doppler étendu du fouillis autour de la vitesse nulle et à toutes les cases distances.

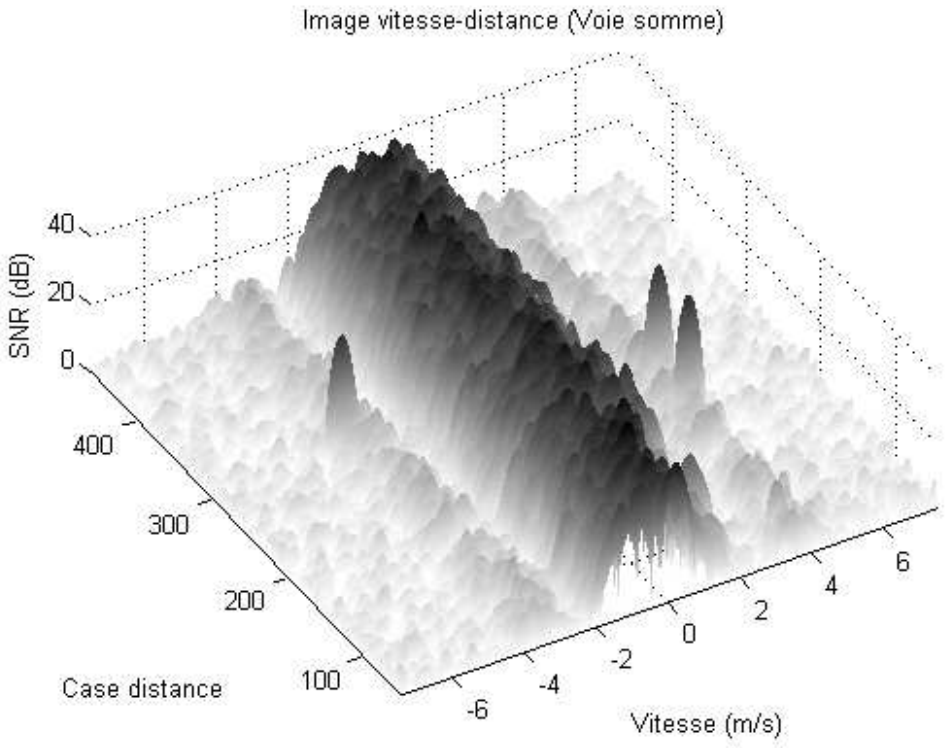


FIGURE 1.3 – Carte distance-vitesse de la voie somme classique

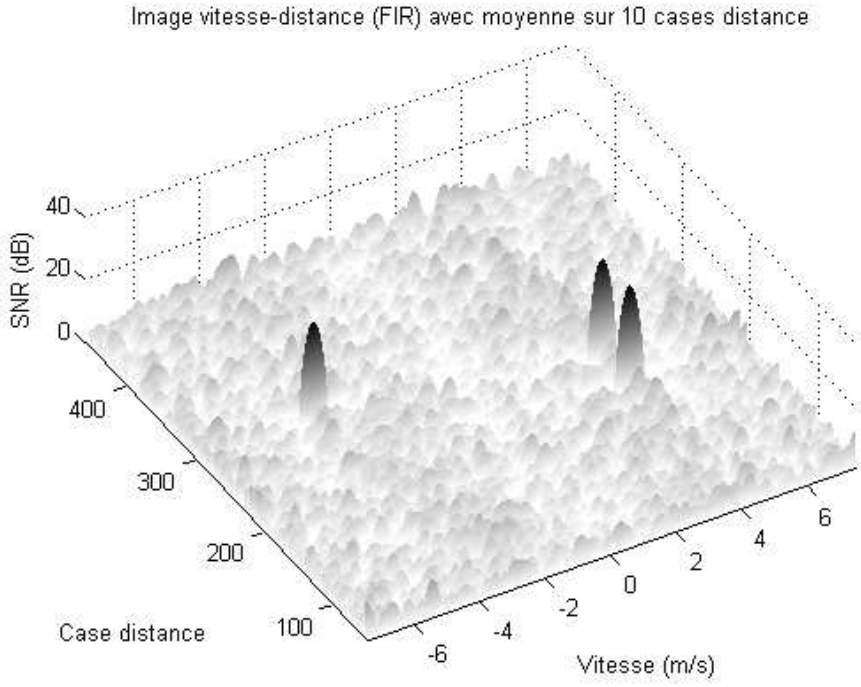


FIGURE 1.4 – Filtre FIR de référence avec moyenne sur 10 cases distance

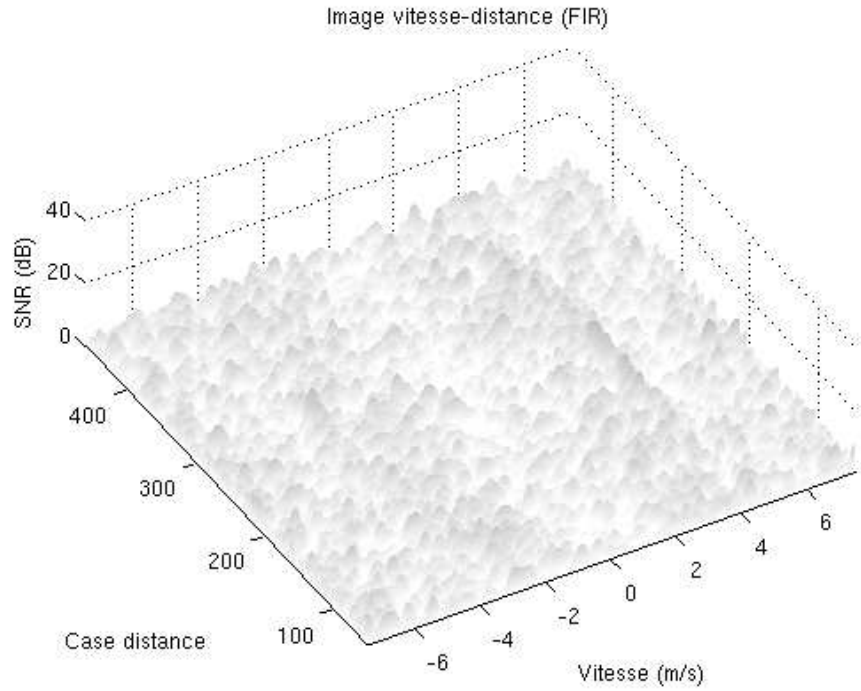


FIGURE 1.5 – Filtre FIR exploitant uniquement les donnée dans la CST

Les résultats du traitement FIR standard sont présentés en Figure 1.4, avec une moyenne sur 10 cases distance pour l'estimation de la matrice et deux cases de garde autour de la case sous test (exclusion du signal utile des données d'apprentissage). Le paramétrage retenu pour les filtrages STAP est $M = 4$, $M_p = 64$, $K_t = 61$. Sur ces données globalement à fouillis homogène et faible densité de cibles, les filtres STAP FIR se comportent de manière tout à fait satisfaisante. Le fouillis présent autour des vitesses nulles est correctement filtré, et les cibles sont bien préservées.

La Figure 1.5 correspond à la mise en œuvre de ce même filtre FIR mais en exploitant seulement les données de la CST, stratégie à laquelle on serait conduit sur des données fortement hétérogènes. On constate bien, comme attendu, la suppression des signaux utiles par le filtrage.

La Figure 1.6 correspond à la mise en œuvre du détecteur MLED standard dans la même situation. On constate cette fois que les cibles sont bien préservées et le fouillis éliminé. On note également la finesse du pic de détection, même avec le sur-échantillonnage par un facteur 4 de l'analyse Doppler utilisée ici. La Figure 1.7 correspond quant à elle à la mise en œuvre du traitement Stop-Band APES dans la même situation. Là encore, les cibles sont bien préservées et le fouillis éliminé. Comme attendu, la finesse du pic de détection est bien moindre par rapport au détecteur MLED. La Figure 1.8, la Figure 1.9, la Figure 1.10, et la Figure 1.11, correspondent à la mise en œuvre des traitements MLED et Stop-Band APES sans sur-échantillonnage de l'analyse Doppler, pour différentes positions en Doppler des cibles au sein d'une case de résolution : cibles décalées d'un quart de case ou d'une demi-case par rapport à la grille d'échantillonnage Doppler. Le traitement MLED standard ne permet pas la détection des cibles, même lorsque celles-ci sont décalées d'un quart de case. Ceci est la conséquence de la finesse du projecteur MLED identifiée en Section 1.1. En revanche, cette détection n'est pas détériorée avec le détecteur Stop-Band APES, même pour le cas des cibles décalées d'une demi-case.

On représente ensuite en Figure 1.12, Figure 1.13, et Figure 1.14 les pertes en SNIR, représentées en valeur positive sur les figures 3D lorsqu'il y a une perte pour une meilleure visibilité, par rapport au traitement en environnement clair (i.e. sans fouillis, bruit thermique seulement présent). On constate une détection quasiment sans perte jusqu'à une vitesse minimale de 1 $m.s^{-1}$, contre au mieux 2 $m.s^{-1}$ sur la voie somme classique (cf Figure 1.3), en supposant maîtrisée de plus la fausse alarme au pied du lobe principal de fouillis. L'entaille Doppler au niveau des cibles pour les traitements MLED et Stop-Band APES est représentative de l'exclusion du signal cible par les projecteurs identifiés dans la section précédente.

La largeur et la profondeur de cette entaille, caractéristiques des différents traitements, peuvent être évaluées plus précisément sur la Figure 1.15. Cette figure fournit une représentation classique des pertes SNIR des différents traitements pour la case distance 296 où sont représentées les coupes superposées et inversées, pour la case distance 296, des pertes SNIR présentées en Figure 1.12, Figure 1.13, et Figure 1.14.

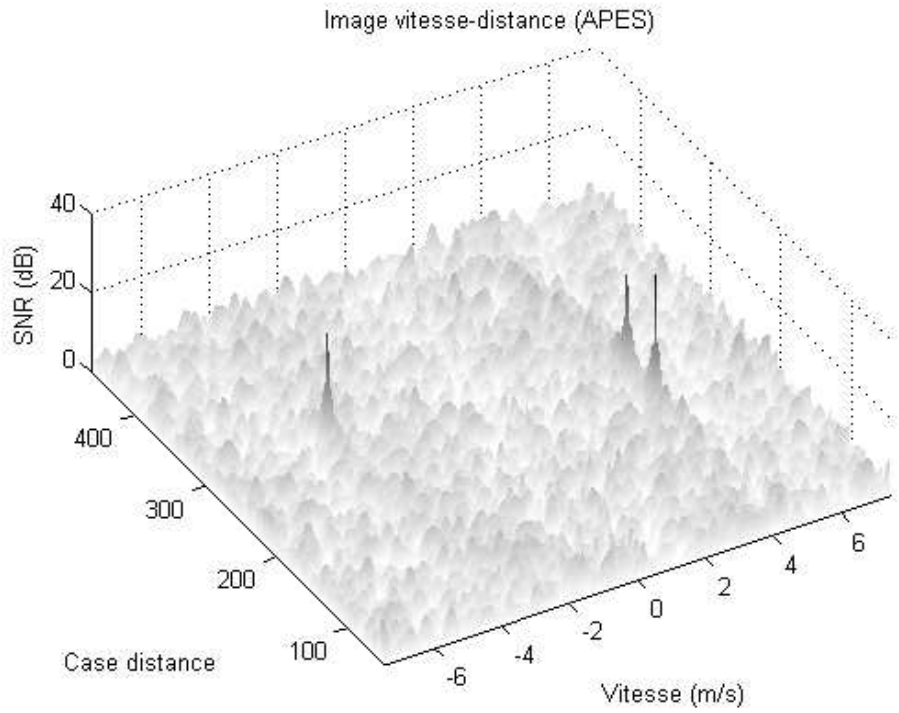


FIGURE 1.6 – Détecteur MLED standard

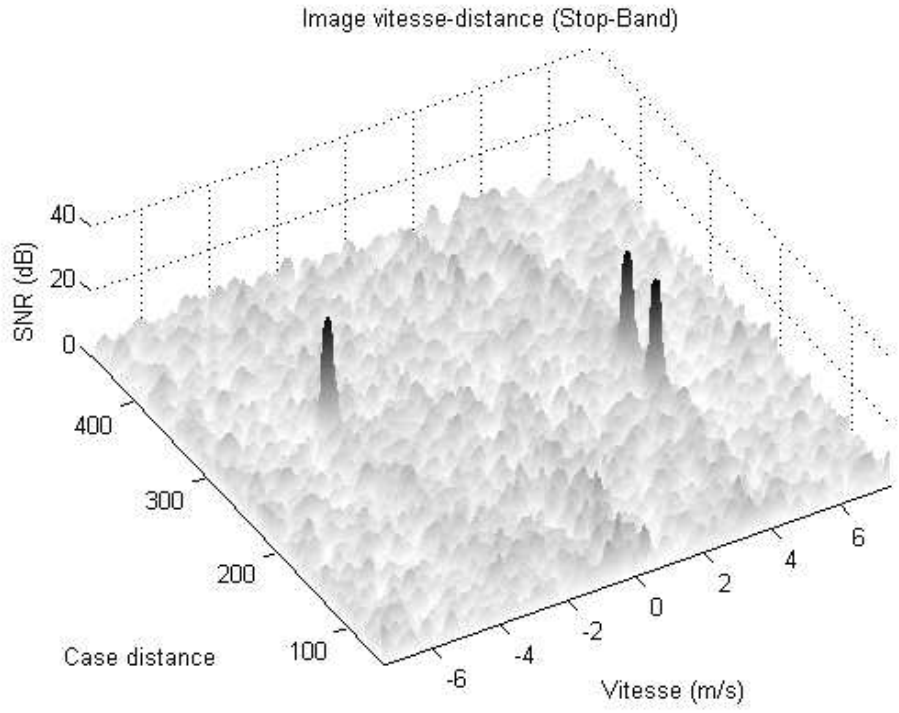


FIGURE 1.7 – Détecteur Stop-Band APES

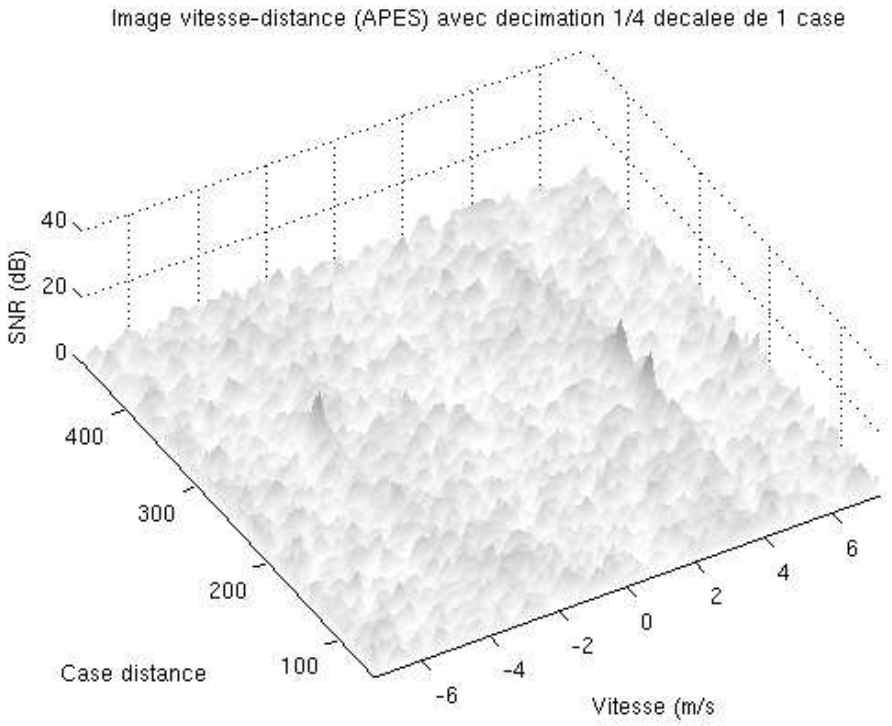


FIGURE 1.8 – Détecteur MLED sans sur-éch. – cibles décalées d'1/4 de case

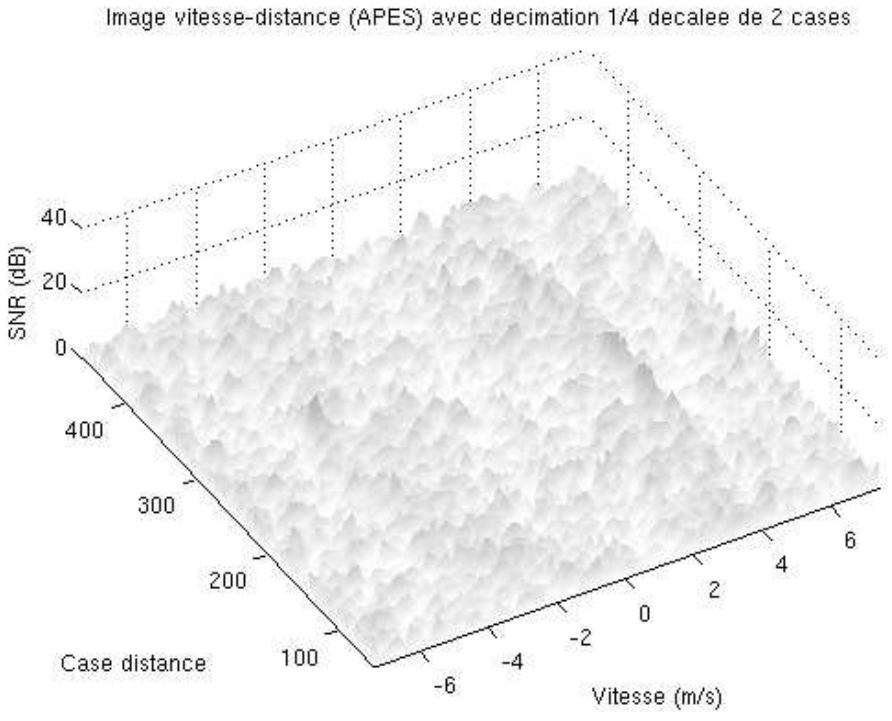


FIGURE 1.9 – Détecteur MLED sans sur-éch. – cibles décalées d'1/2 case

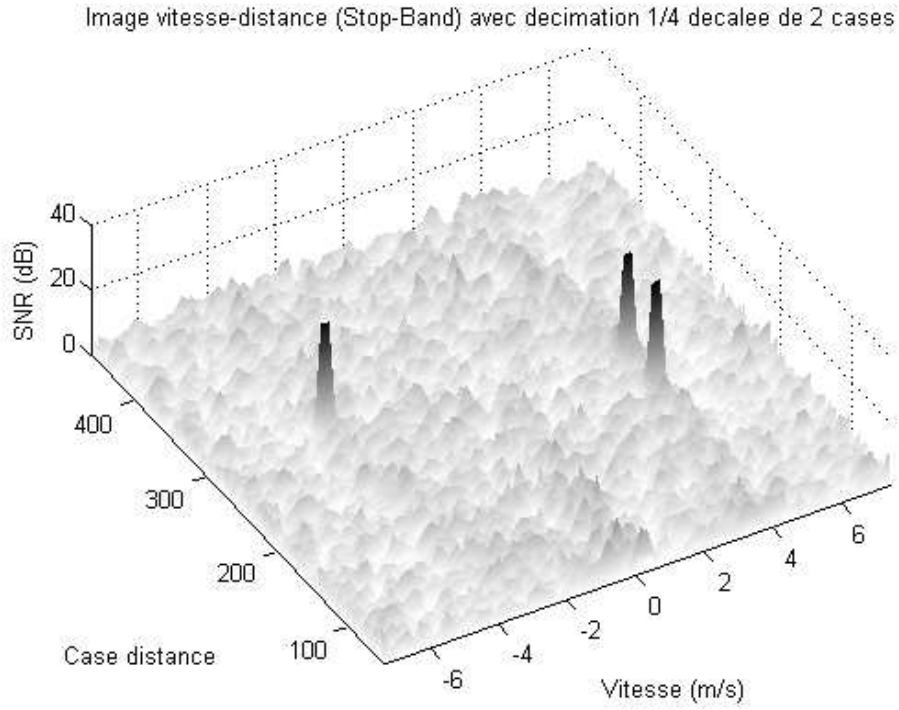


FIGURE 1.10 – Détecteur Stop-Band sans sur-éch. – cibles décalées d'1/4 de case

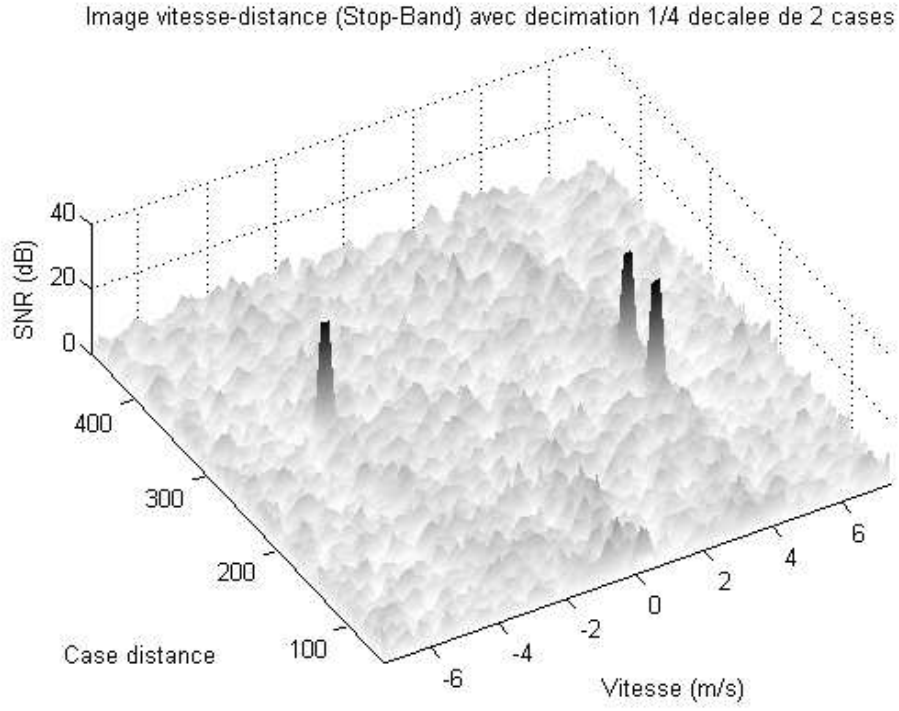


FIGURE 1.11 – Détecteur Stop-Band sans sur-éch. – cibles décalées d'1/2 case

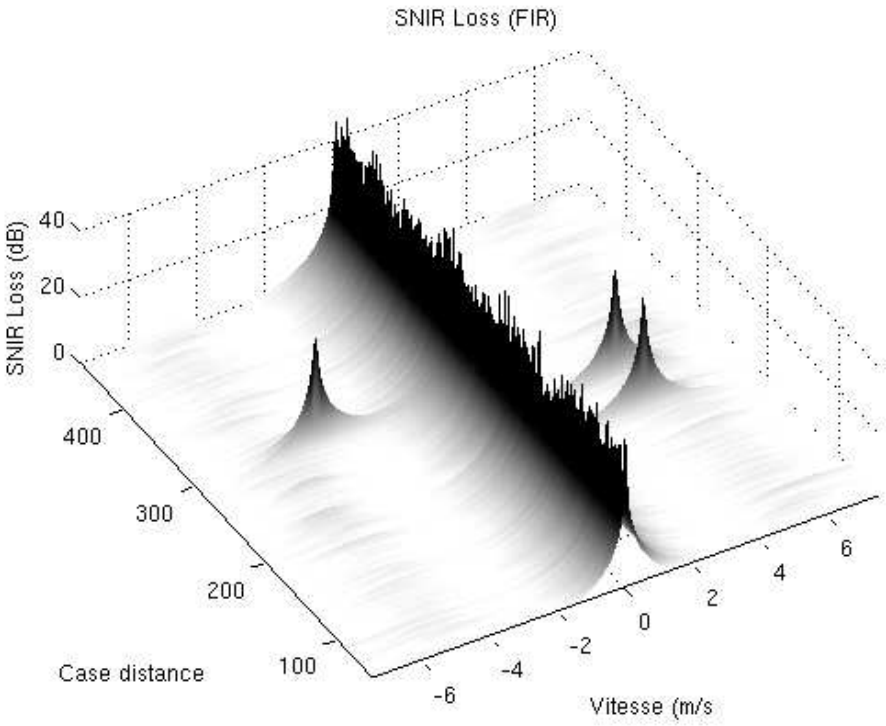


FIGURE 1.12 – Perte SNIR avec filtre FIR exploitant les seules donnée de la CST

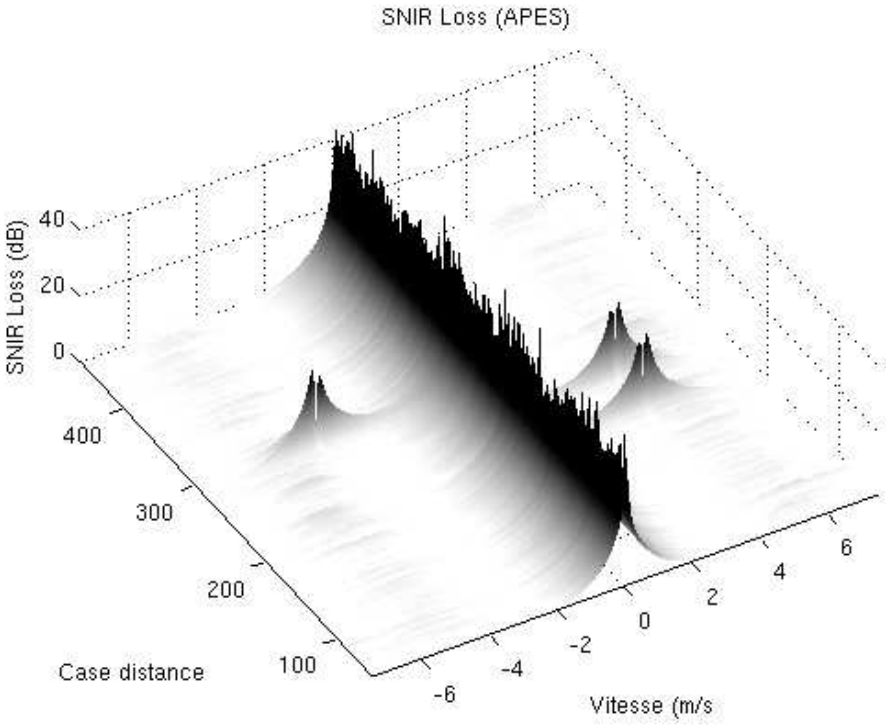


FIGURE 1.13 – Perte SNIR avec le détecteur MLED

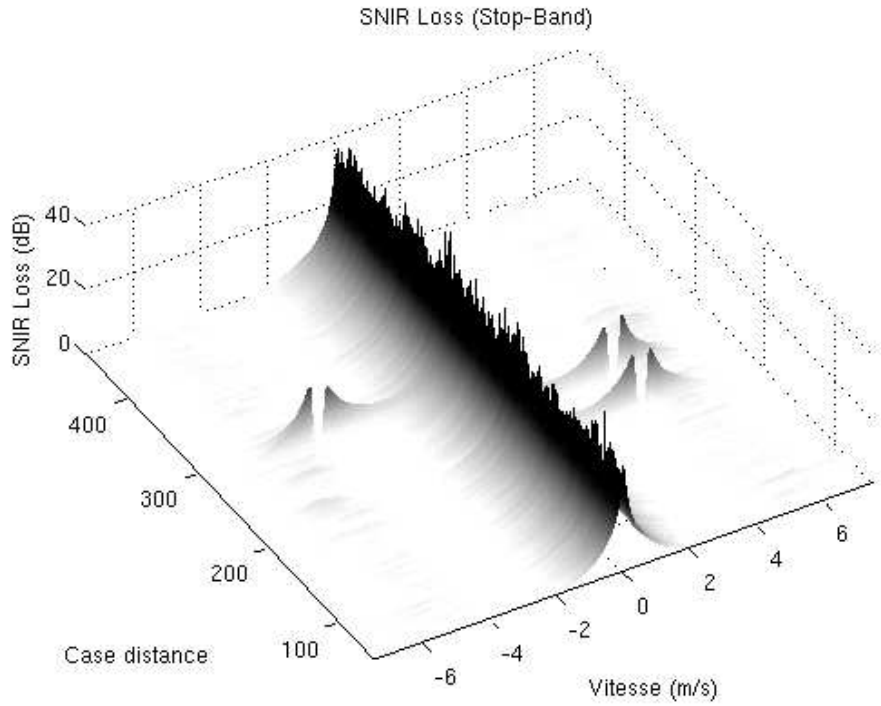


FIGURE 1.14 – Perte SNIR avec le détecteur Stop-Band APES

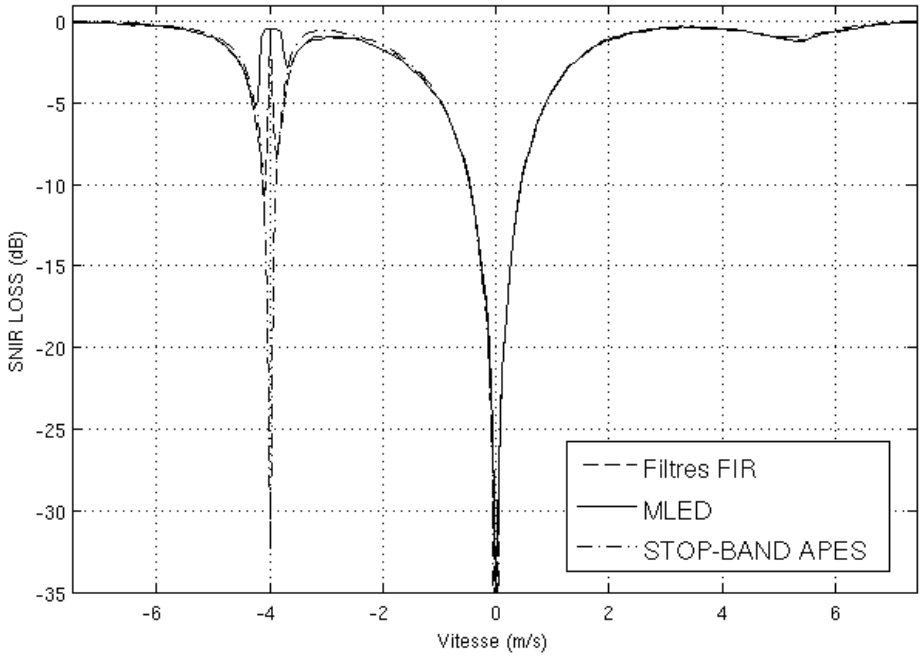


FIGURE 1.15 – SNIR LOSS pour la case distance 296. La cible se trouve à $-4\ m.s^{-1}$.

1.4.2 Résultats sur données ONERA

Les résultats suivants ont été obtenus sans sur-échantillonnage sur les données **ONERA-AS** (voir Annexe A.3.1). Comme pour les données précédentes, les matrices de covariance sont estimées sur une case distance sauf pour le traitement référence (filtre FIR) où elles sont estimées sur 10 cases. Le paramétrage retenu pour les filtrages STAP est $M = 4$, $M_p = 64$, $K_t = 61$.
Sur la Figure 1.16, on peut observer la sortie du traitement Doppler classique de la voie somme habituelle. On peut noter la présence de pics au niveau des nombreuses cibles, ainsi que le spectre Doppler étendu du fouillis autour de la vitesse nulle et à toutes les cases distance. Sur cette figure, le triangle repère une zone dense en cibles, le carré un convoi, et l'ovale du fouillis piqué.
La Figure 1.17 montre que le traitement référence supprime efficacement le fouillis. Cependant, à cause de l'utilisation de cases distance voisines pour l'estimation de la matrice de covariance, les cibles du convoi ainsi que les cibles situées en zone très dense sont atténuées de manière importante. Par rapport à MLED (Figure 1.18), la nouvelle méthode évite la suppression des cibles se trouvant entre 2 cases Doppler lorsqu'il n'y a pas de sur-échantillonnage. N'utilisant pas de données secondaires, le filtre Stop-Band APES ne supprime ni le convoi, ni les cibles situées en zone dense (Figure 1.19) mais n'arrive pas à supprimer correctement le fouillis piqué. Ce phénomène est lié à l'étendue du projecteur Stop-Band APES qui, couplé à la forte puissance du fouillis dans cette zone, n'arrive pas à correctement atténuer le fouillis. Nous étudierons cette limitation dans le Chapitre 3.

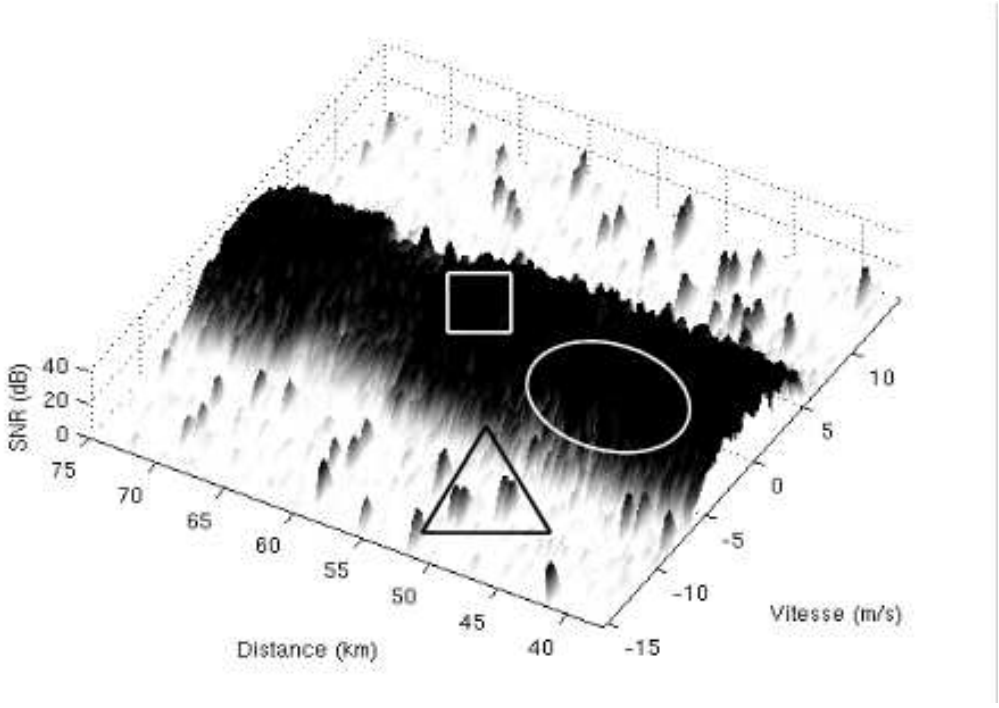


FIGURE 1.16 – Carte vitesse-distance de la voie somme classique

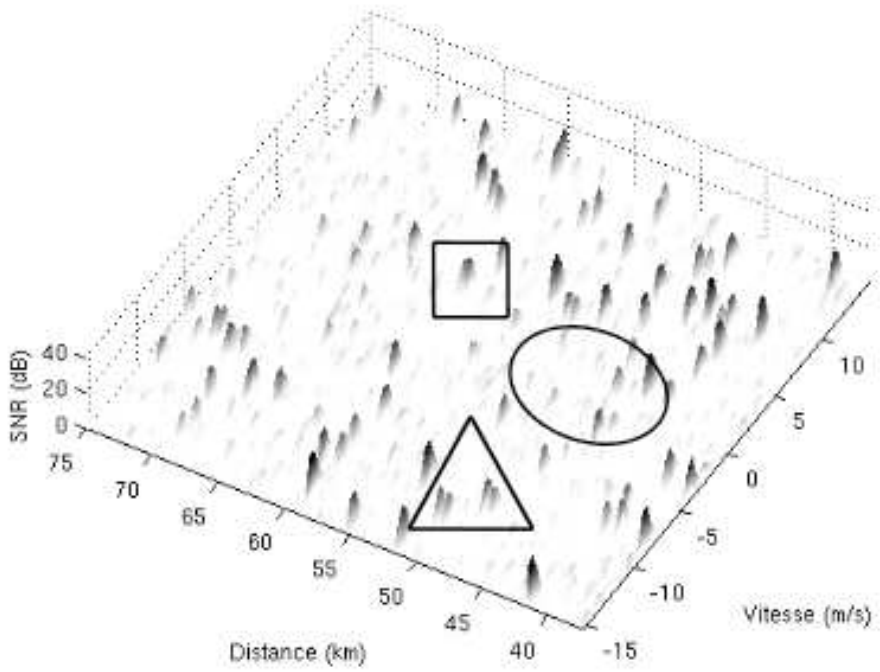


FIGURE 1.17 – Filtre FIR de référence avec moyenne sur 10 cases distance

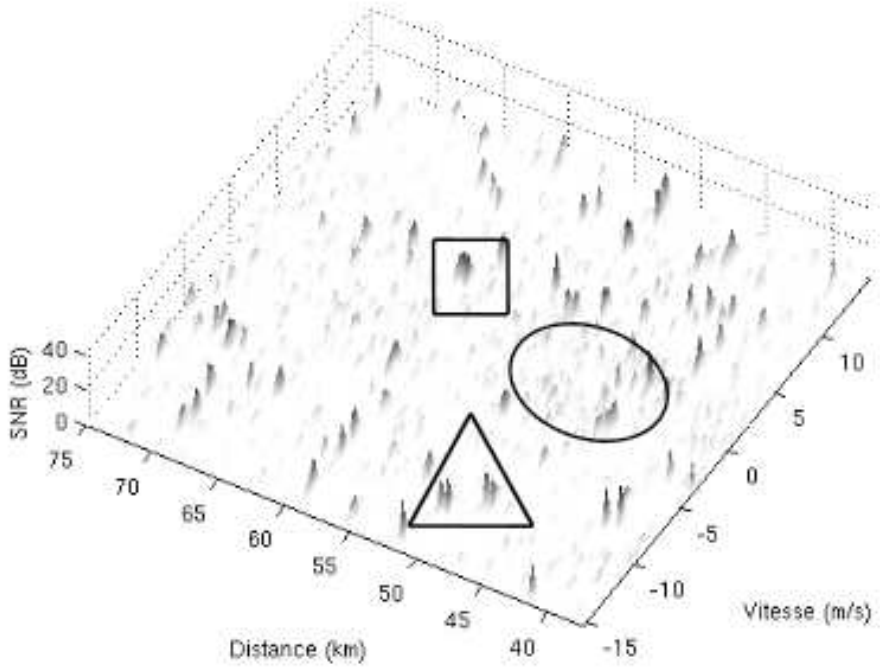


FIGURE 1.18 – Détecteur MLED sur données primaires seules

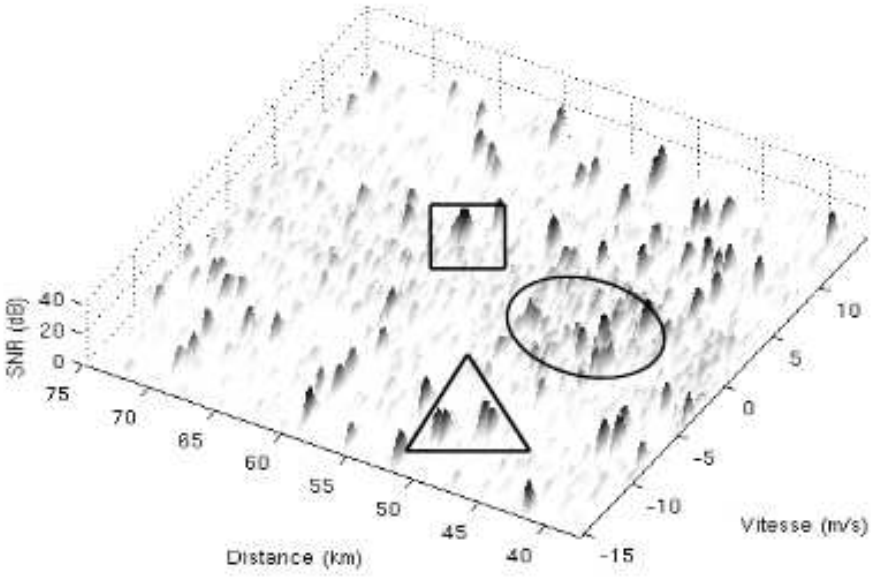


FIGURE 1.19 – Détecteur Stop-Band sur données primaires seules

1.5 Conclusion

Nous avons présenté dans ce chapitre une approche alternative aux traitements adaptatifs usuels, visant à n'exploiter que les données de la case sous test, sans rechercher des données d'apprentissage ne contenant pas le signal utile. Cette approche est particulièrement bien adaptée au cas d'environnements fortement hétérogènes (fouillis non stationnaire en distance, forte densité de cibles), dans lesquels la représentativité des données d'apprentissage est prise en défaut. Le traitement proposé, nommé Stop-Band APES, est une extension du détecteur MLED, interprété comme une minimisation de l'énergie en dehors de la bande du signal utile, qui permet de réduire considérablement la charge de calcul de ce dernier en supprimant ses propriétés intrinsèques d'hyper-résolution Doppler, non recherchées en STAP. L'application de la méthode proposée, appelée « Stop-Band APES » a démontré son efficacité sur des données synthétiques. Dans certaines situations où le nombre d'échantillons est limité, le détecteur Stop-Band APES, tout comme le MLED et le traitement FIR de référence qui sont tous basés sur l'inversion de la matrice de covariance, verra ses performances chuter. Dans le Chapitre suivant, nous étudions ce phénomène et nous proposons des méthodes visant à surmonter ces problèmes liés à la mauvaise estimation de la matrice de covariance.

2

Estimation de la matrice de covariance

Sommaire

Les méthodes opérant sur les seules données primaires, si elles ont l'avantage d'être robustes face à l'hétérogénéité du fouillis, utilisent un algorithme d'inversion matricielle pour construire le filtre adapté. Elles peuvent donc être sévèrement impactées par la mauvaise estimation de la matrice de covariance. En effet, pour obtenir des performances satisfaisantes, en se restreignant à ces seules données, le nombre de données d'entrainement doit dans ce cas être supérieur à $2NM$. Dans ce chapitre, nous démontrons que nous pouvons utiliser des méthodes STAP à rang réduit avec des algorithmes à données primaires seules pour améliorer les performances des détecteurs. Nous introduisons aussi un algorithme qui réduit la charge de calcul par rapport aux méthodes de sous-espace traditionnelles basées sur la décomposition en éléments propres. Les résultats sur données synthétiques réalistes montrent que cette approche améliore sensiblement la réjection du fouillis.

2.1 Régularisation de la matrice de covariance

2.1.1 Projections alternées

La méthode la plus simple et la plus utilisée pour résoudre les problèmes liés à la mauvaise estimation de la matrice de covariance est appelée *Diagonal Loading* (voir Annexe B.4). Ici, nous explorons une première voie pour résoudre ces problèmes en essayant d'exploiter certaines propriétés des matrices de covariance. En effet, la vraie matrice de covariance $\mathbf{\Gamma}$ est Hermitienne définie positive et possède une structure Toeplitz bloc-Toeplitz. La méthode des projections alternées permet de régulariser la matrice de covariance estimée $\mathbf{R}$.

Soit $\mathbf{T_1}$ et $\mathbf{T_2}$ deux matrices Toeplitz, pour $0 \leq \alpha \leq 1$, alors $\alpha\mathbf{T_1} + (1-\alpha)\mathbf{T_2}$ est aussi Toeplitz donc l'ensemble de toutes les matrices Toeplitz forme un espace convexe fermé C_1. De même, si $\mathbf{R_1}$ et $\mathbf{R_2}$ sont deux matrices définies positives, alors $\alpha\mathbf{R_1}+(1-\alpha)\mathbf{R_2}$ est définie positive donc l'ensemble des matrices définies positives forme un ensemble convexe fermé. Tout ensemble convexe C possède un opérateur de projection associé $\mathbf{P}$ tel que pour tout vecteur $\mathbf{x}$, il existe un unique vecteur $\mathbf{y} = \mathbf{Px}$ qui appartient à C tel que $\| \mathbf{x} - \mathbf{Px} \|$ est minimum sur C. Le vecteur $\mathbf{Px}$ est la projection de $\mathbf{x}$ sur l'ensemble convexe C [42]. Les deux opérateurs de projection $\mathbf{P_1}$ et $\mathbf{P_2}$ sur les deux ensembles convexes sont bien connus [43] et détaillés ci-dessous :

Projection sur l'ensemble des matrices Toeplitz :

Soit $\mathbf{R} = (R_{ij})$ une matrice bloc de taille $MN \times MN$, et $\mathbf{T} = \mathbf{P_1R}$ sa projection Toeplitz sur C_1, on a :

$$\mathbf{T} = \mathbf{P_1R} = \begin{bmatrix} T_0 & T_1 & \cdots & T_n \\ T_1^* & T_0 & \cdots & T_{n-1} \\ \vdots & \vdots & \ddots & \cdots \\ T_n^* & T_{n-1}^* & \cdots & T_0 \end{bmatrix} \tag{2.1}$$

avec

$$T_k = \frac{1}{n-k} \sum_{i=1}^{n-k} R_{i,i+k} \tag{2.2}$$

Projection sur l'ensemble des matrices définies positives :

De même, la projection sur l'ensemble C_2 des matrices définies positives de :

$$\mathbf{R} = \mathbf{V\Lambda V^H} \tag{2.3}$$

est la matrice suivante :

$$\mathbf{R} = \mathbf{P_2R} = \mathbf{V\Lambda_+ V^H}, \tag{2.4}$$

où

$$(\Lambda_+)_{i,i} = \begin{cases} \lambda_i, & \text{si } \lambda_i > 0 \\ 0, & \text{si } \lambda_i \leq 0 \end{cases} \tag{2.5}$$

Si l'on connait la valeur propre minimum de $\mathbf{R}$, dans notre cas elle correspond au seuil du niveau de bruit thermique σ^2, alors :

$$(\Lambda_+)_{i,i} = \begin{cases} \lambda_i, & \text{si } \lambda_i > \sigma^2 \\ \sigma^2, & \text{si } \lambda_i \leq \sigma^2 \end{cases} \tag{2.6}$$

La méthode des projections alternées applique ces opérateurs de projection tour à tour tel qu'à la n-ième itération nous avons :

$$\mathbf{R_n} = \mathbf{P_2 P_1 R_{n-1}} \tag{2.7}$$

Ces projections alternées convergent au moins faiblement vers un élément de l'intersection des deux ensembles C_1 et C_2 [44]. Après plusieurs itérations, la matrice de covariance possèdera les deux propriétes voulues.

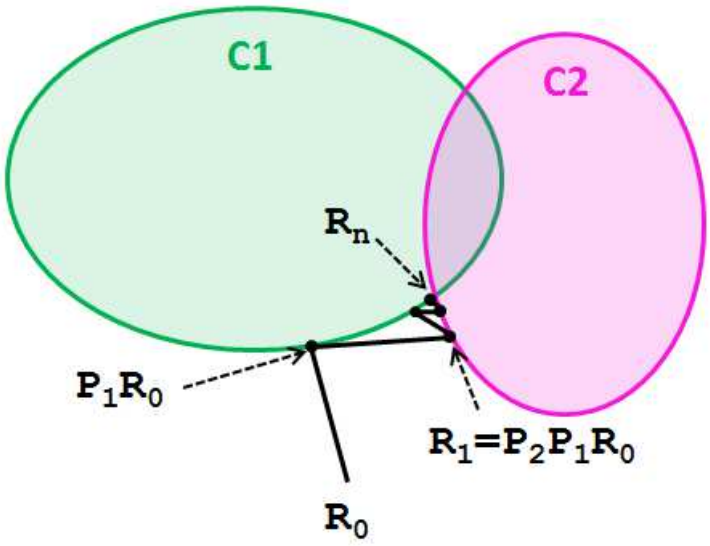


FIGURE 2.1 – Schéma de la méthode des projections alternées de $\mathbf{R_0}$ sur deux ensembles convexes C_1 et C_2

La méthode *Relaxed projection operator* introduite dans [44] est une forme semblable à celle des projections alternées mais ajoute un coefficient de pondération λ tel que la projection sur l'ensemble convexe C_k s'écrive sous la forme suivante, pour m espaces convexes :

$$\mathbf{Q_k} = \mathbf{I} + \lambda_k(\mathbf{P_k} - \mathbf{I}), \;\; 0 < \lambda_k < 2, \;\; k = 1, 2, ..., m \tag{2.8}$$

L'opérateur de projection $\mathbf{P_k}$ est toujours le même, et λ_k est un paramètre qui doit être choisi dans l'ajustement de la pondération de l'opérateur de projection. L'équation (2.7) devient alors :

$$\mathbf{R_n} = \mathbf{Q R_{n-1}} = \mathbf{Q_2 Q_1 R_{n-1}} \tag{2.9}$$

et $\mathbf{Q^n R_0}$ converge au moins faiblement vers un point de l'intersection $C_0 = \bigcap_{i=1}^{m} C_i$, c'est à dire dans notre cas $C_1 \cap C_0$.

Le paramètre de pondération λ_k apporte à cette méthode plusieurs avantages par rapport à la méthode des projection alternées classique. Ce paramètre permet d'accélerer la convergence de l'algorithme. De plus, après chaque itération, la nouvelle matrice peut se trouver entièrement dans un espace convexe dans le cas où $1 < \lambda_k < 2$ ce qui rend son estimation plus robuste.

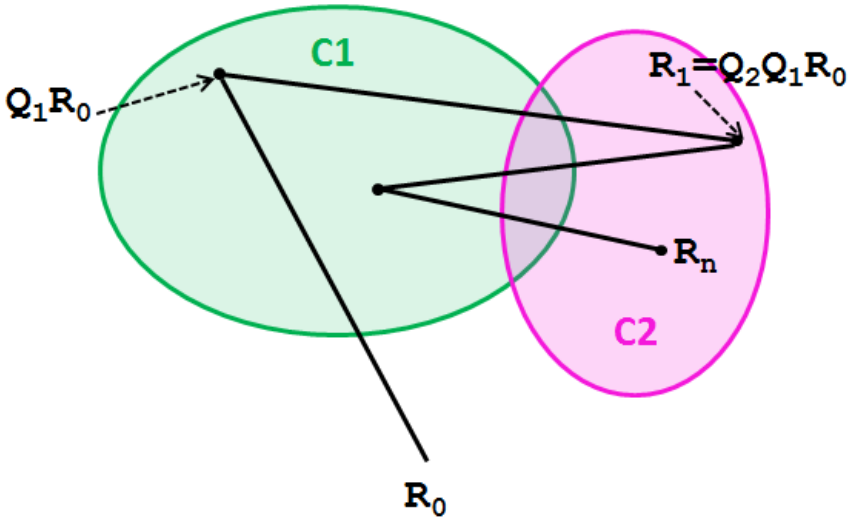


FIGURE 2.2 – Représentation de la méthode *Relaxed projection operator* de x_0 sur deux ensembles convexes C_1 et C_2 avec un paramètre $1 < \lambda_k < 2$

Nous pouvons étudier la convergence de cette méthode. Par exemple, dans le cas où on ne projette que sur C_1, prenons $\mathbf{P_1}$ l'opérateur de projection dans C_1. On a :

$$\mathbf{R_n} = (1-\lambda)\mathbf{R_{n-1}} + \lambda \mathbf{P_1 R_{n-1}} \tag{2.10}$$

soit pour les itérations suivantes :

$$\mathbf{R_1} = (1-\lambda)\mathbf{R_0} + \lambda \mathbf{P_1 R_0}$$

$$\mathbf{R_2} = (1-\lambda)[(1-\lambda)\mathbf{R_0} + \lambda \mathbf{P_1 R_0}] + \lambda \mathbf{P_1}[(\mathbf{1}-\lambda)\mathbf{R_0} + \lambda \mathbf{P_1 R_0}]$$

...

$$\mathbf{R_n} = (1-\lambda)^n \mathbf{R_0} + \sum_{i=1}^{n} (1-\lambda)^{i-1} \lambda \mathbf{P_1 R_{n-i}} \tag{2.11}$$

Si $0 < \lambda < 1$ alors $(1-\lambda)^n \mathbf{R_0} \longrightarrow 0$ et $\mathbf{R_n}$ est une somme de matrices Toeplitz donc est Toeplitz.
Si $1 < \lambda < 2$, le terme $(1-\lambda)^n \mathbf{R_0} \longrightarrow 0$ et la matrice converge dans C_1. Pour le cas $\lambda = 1$, nous revenons à la méthode précédente, c'est à dire que $\mathbf{Q_k} = \mathbf{P_k}$.

Résultats

Nous étudions ici les performances de la méthode des projections alternées sur les données **SimALU** (voir (voir Annexe A.3.3). Nous avons choisi les paramètres suivants : antenne ALU 8 voies à visée latérale, $M_p = 32$ impulsions, la vitesse de la plateforme est $100m/s$. Une cible de vitesse $30m/s$ a été ajoutée à la case distance 20. La vitesse de la cible a été choisie de façon à ce qu'elle soit endo-clutter, c'est à dire non détectable sans traitement STAP. Nous fixons la dimension temporelle du filtre STAP à $M = 6$, le vecteur est de taille 48. Le fouillis étant homogène en distance, nous pouvons utiliser un grand nombre de cases distance pour l'estimation de la matrice de covariance de la case distance sous test. Pour évaluer la méthode des projections alternées, nous nous limiterons à deux cases distances pour les données d'entrainement. Rappelons que pour une case distance, le nombre d'échantillons est $K_t = M_p - M + 1$ ce qui donne $K_t = 27$ dans notre cas. Avec deux cases distances, nous avons donc 54 estimations pour une taille de vecteur de 48 alors qu'idéalement, le nombre d'estimations doit être au moins égal à 2 fois la taille du vecteur [45]. Comme traitement de référence, nous prendrons un traitement STAP classique, sans *Diagonal Loading* avec une estimation sur 20 cases distances pour nous assurer que la matrice de covariance est estimée sur suffisamment d'échantillons. Les autres traitements n'exploitent que 2 cases distances. Pour les essais avec projections alternées, nous fixons à 10 le nombre d'itérations sur les projections.

Les Figures 2.3 et 2.4 présentent respectivement une image vitesse-distance de la voie somme et une coupe Doppler de la voie somme à la case distance de la cible. Nous voyons que la cible n'est pas détectable sans traitement STAP. Les Figures 2.5 et 2.6 présentent le traitement de référence (avec estimation sur 20 cases distances) où le fouillis est complètement supprimé alors que le signal de la cible est conservé. Comme nous pouvons le voir sur les figures 2.7 et 2.8, le traitement classique sur 2 cases distance sans *Diagonal Loading* ne permet pas la détection de la cible. En revanche, les figures 2.9 et 2.10 montrent que le traitement classique avec *Diagonal Loading* (facteur de chargement $\delta = 1$) permet d'obtenir de bonnes performances, notons que la réjection du fouillis est moins bonne avec *Diagonal Loading* ce qui peut conduire à des fausses alarmes comme nous le verrons dans la Section suivante. Les figures 2.11 à 2.16 montrent les résultats obtenus avec la méthode des projections alternées pour différentes valeurs du paramètre λ. Pour $\lambda = 0.5$, la réjection du fouillis est mauvaise alors que le signal de la cible a été fortement atténué. Avec $\lambda = 1$, le fouillis est parfaitement atténué mais la cible l'est aussi ce qui rend sa détection impossible. Enfin, avec un paramètre $\lambda = 1.6$ le fouillis est sur-annulé tandis que la cible a aussi été supprimée.

Les résultats montrent que si cette méthode permet de rectifier la matrice de manière à correctement supprimer le fouillis, le niveau d'atténuation du signal d'intérêt est tel qu'il rend la détection impossible. De plus, cette méthode est relativement lourde en charge de calcul puisqu'il faut calculer, pour chaque matrice estimée, une décomposition en éléments propres (EVD) à chaque itération afin d'en modifier les valeurs propres. D'autres méthodes de régularisation existent cependant comme la rectification développée dans [46]. Nous allons donc nous intéresser dans ce qui suit à une approche basée sur les sous-espaces fouillis et bruit.

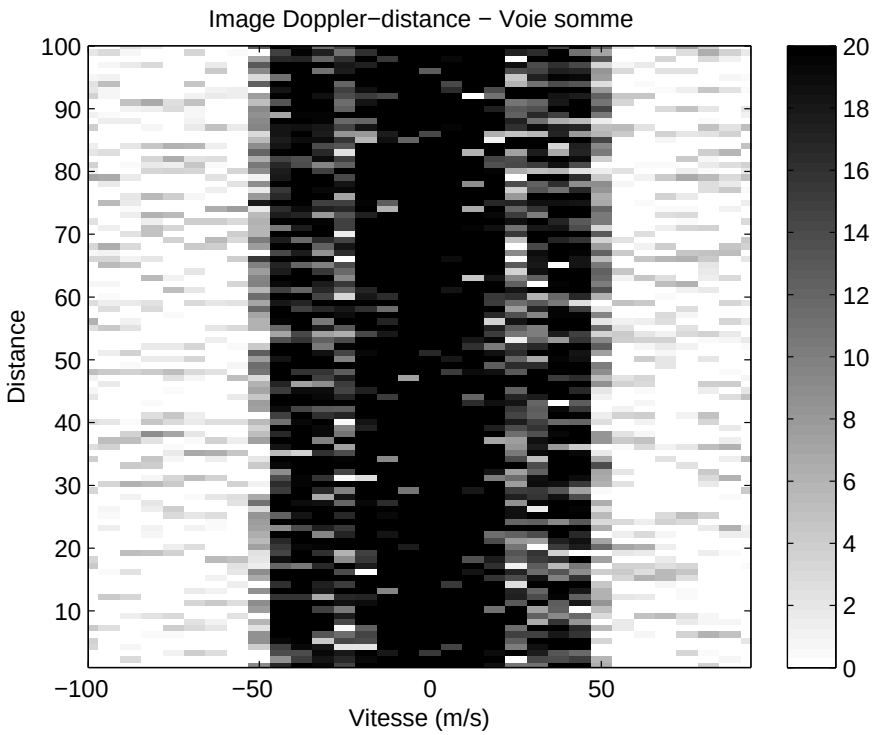


FIGURE 2.3 – Image Doppler-distance de la voie somme

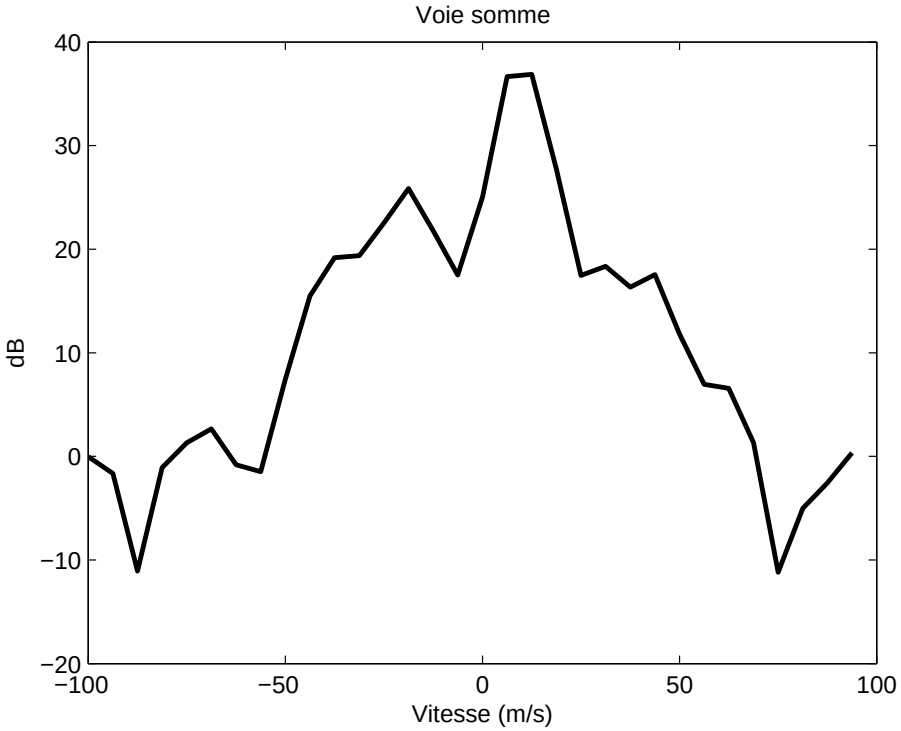


FIGURE 2.4 – SNR en sortie pour la case distance 20 de la voie somme

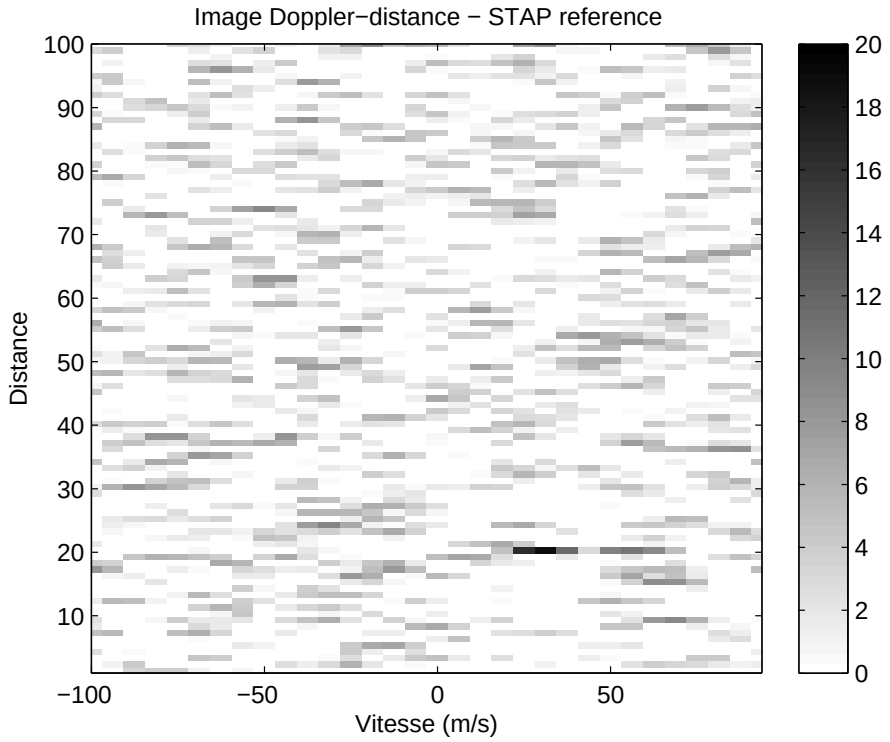


FIGURE 2.5 – Image Doppler-distance du traitement STAP référence

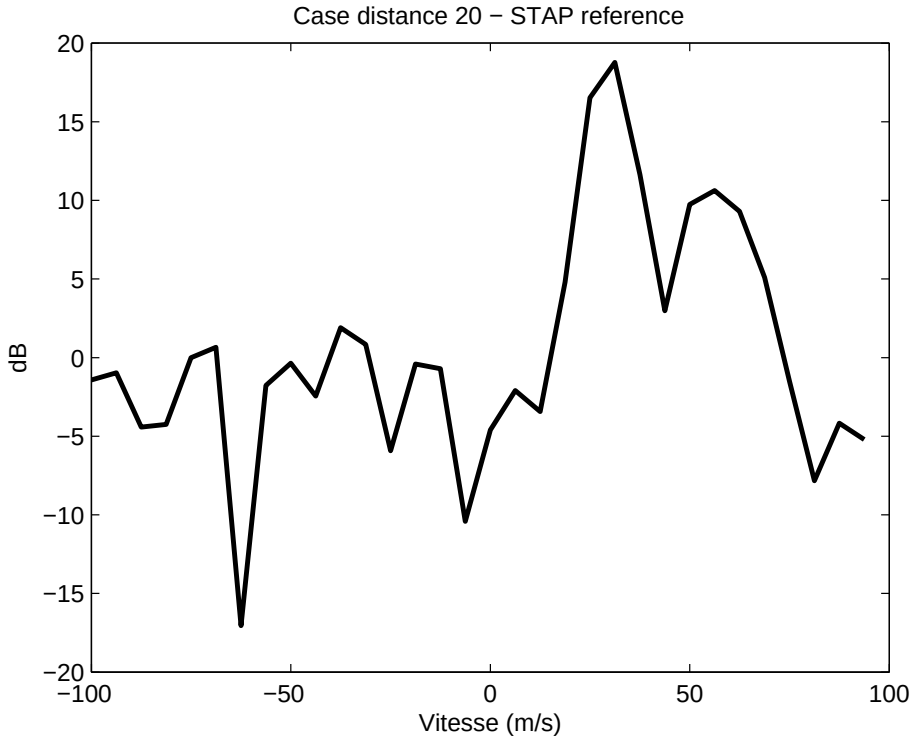


FIGURE 2.6 – SNR en sortie pour la case distance 20 du traitement STAP de référence

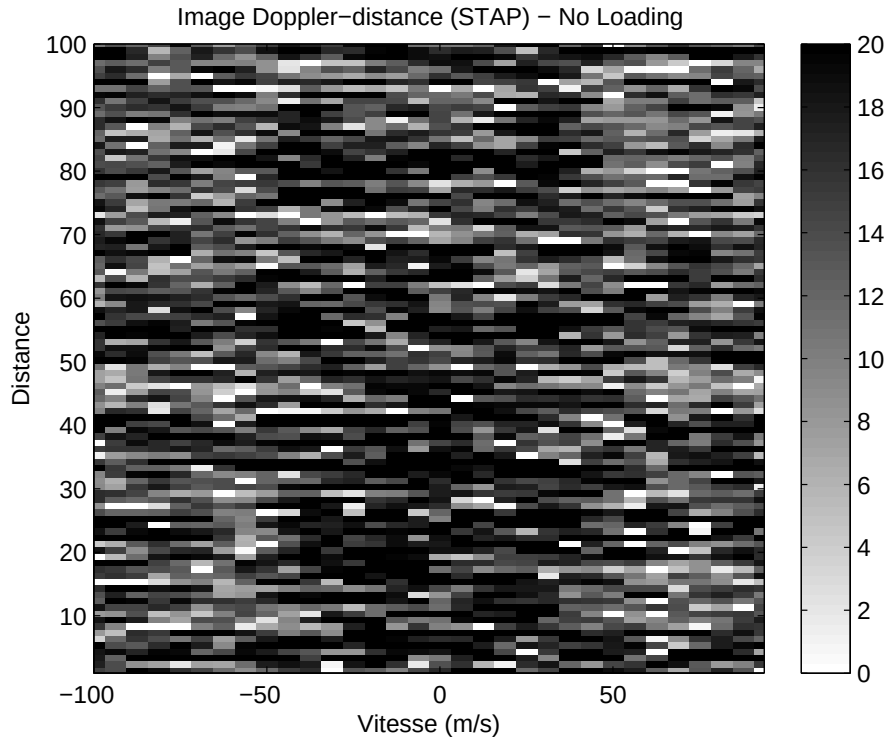


FIGURE 2.7 – Image Doppler-distance du traitement STAP sans Diagonal Loading

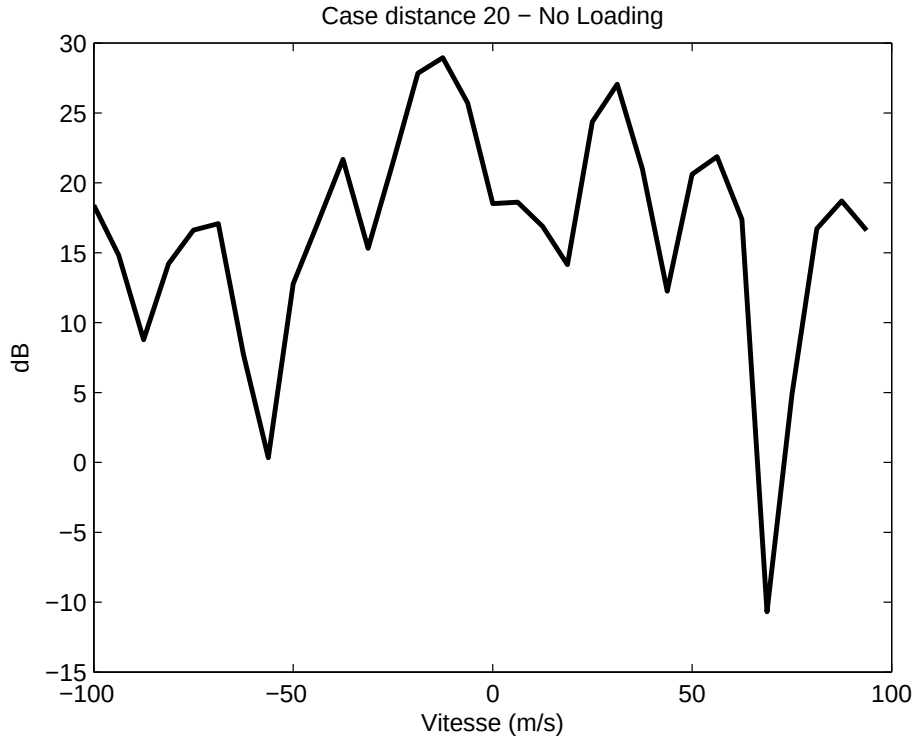


FIGURE 2.8 – SNR en sortie de la case distance 20 sans *Diagonal Loading*

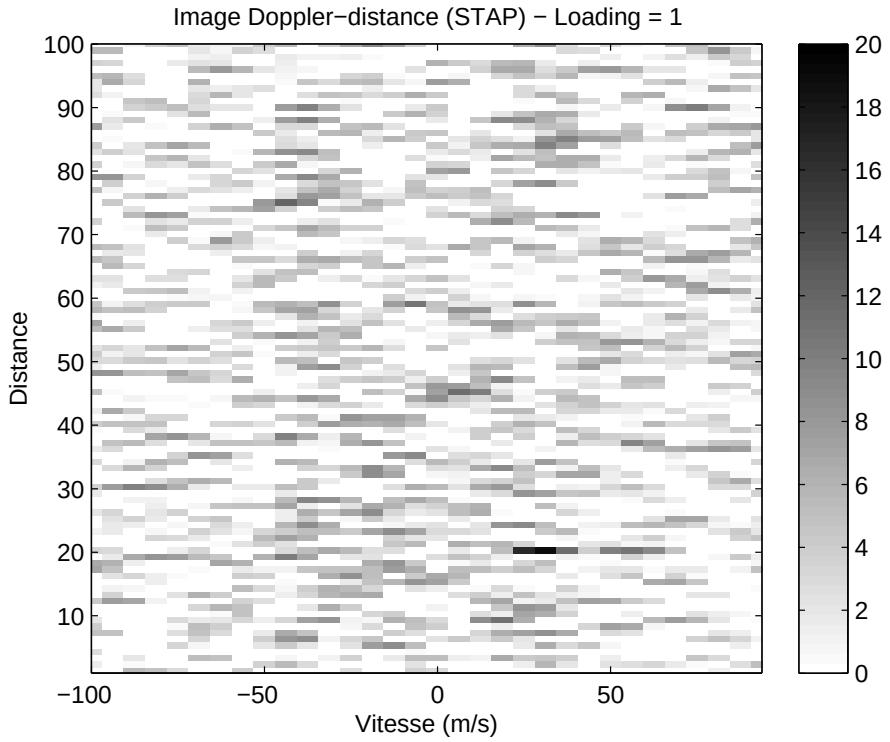


FIGURE 2.9 – Image Doppler-distance du traitement STAP avec *Diagonal Loading*

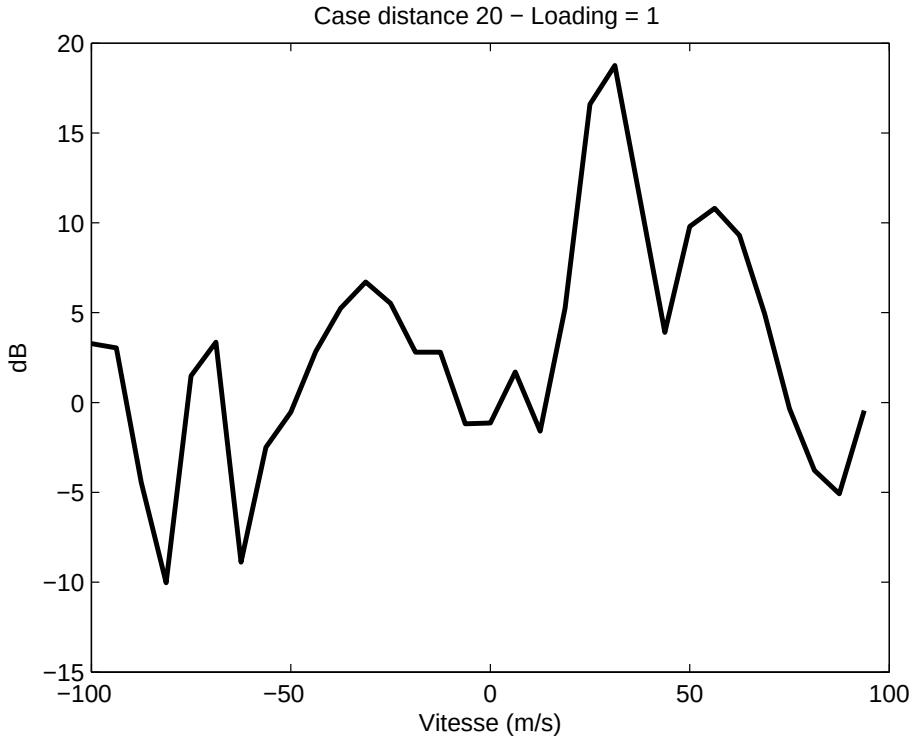


FIGURE 2.10 – SNR en sortie de la case distance 20 avec *Diagonal Loading*

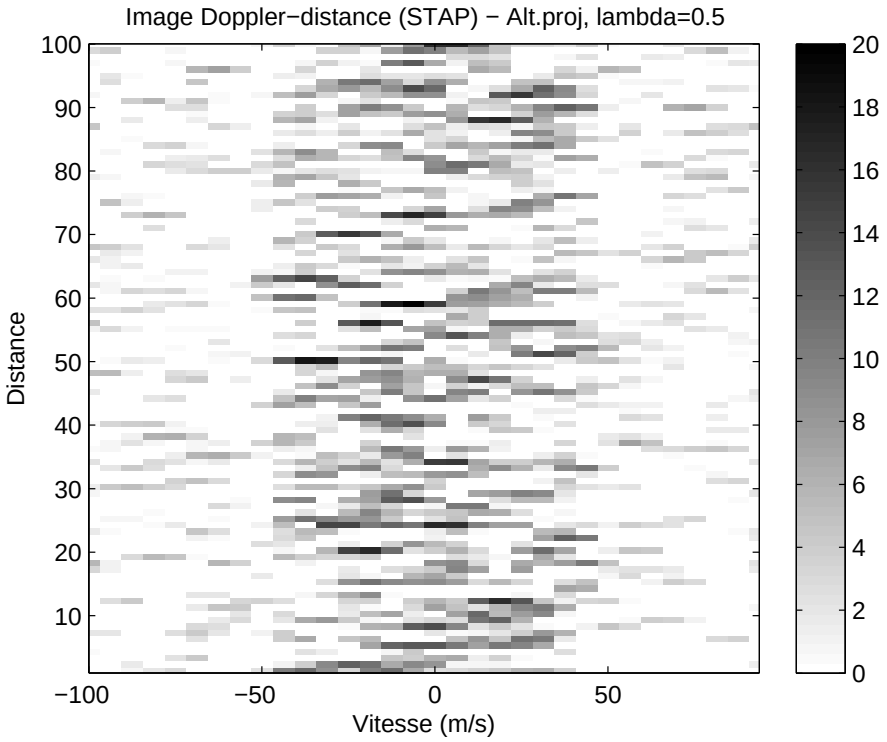


FIGURE 2.11 – Traitement STAP avec projection alternées, Lambda=0.5

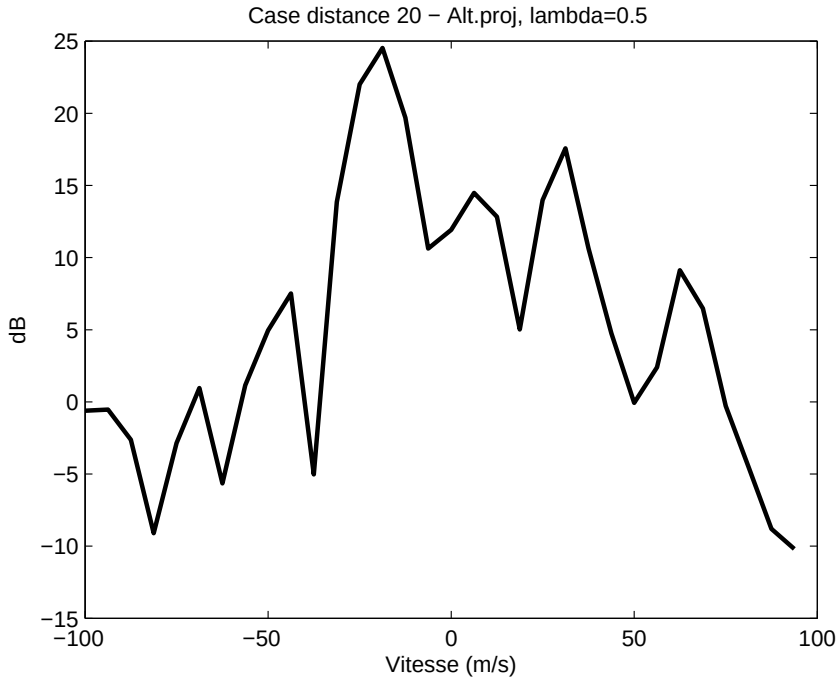


FIGURE 2.12 – Traitement STAP avec projection alternées, Lambda=0.5

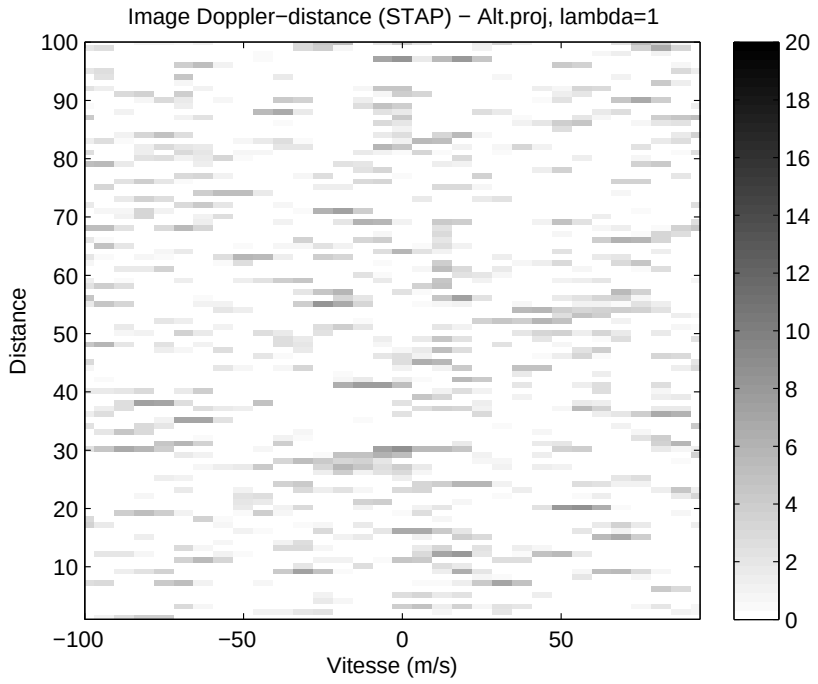


FIGURE 2.13 – Traitement STAP avec projection alternées, Lambda=1

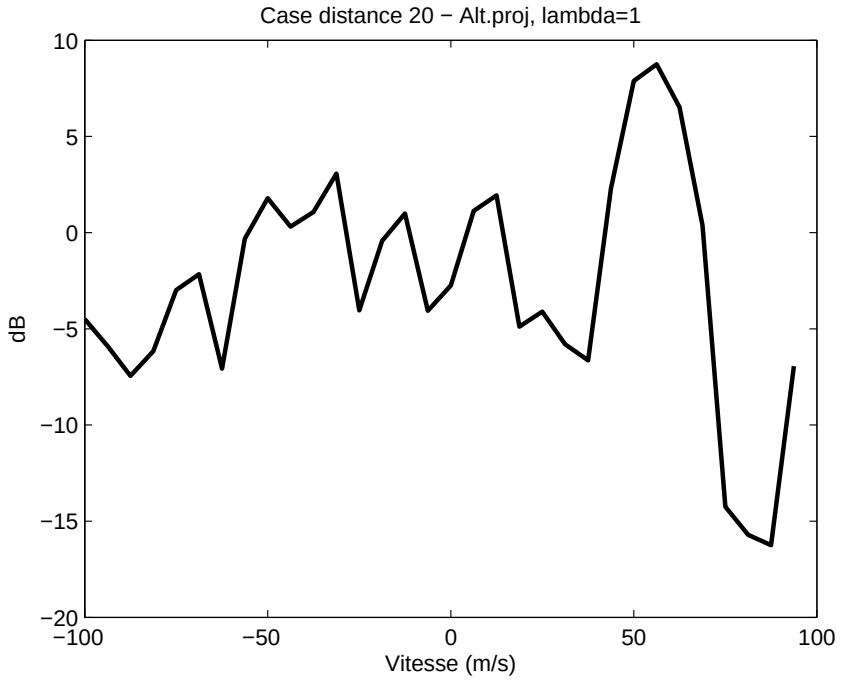


FIGURE 2.14 – Traitement STAP avec projection alternées, Lambda=1

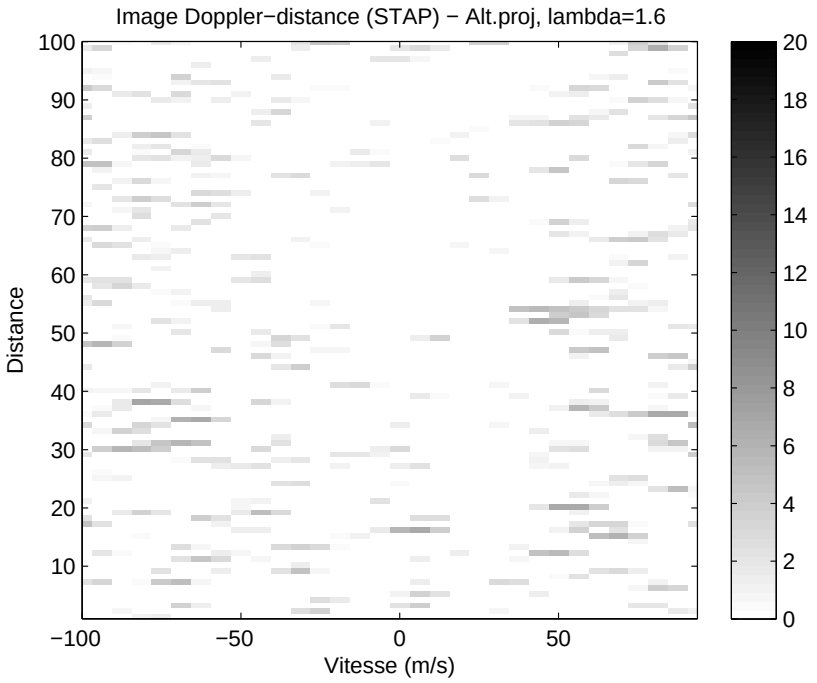


FIGURE 2.15 – Traitement STAP avec projection alternées, Lambda=1.6

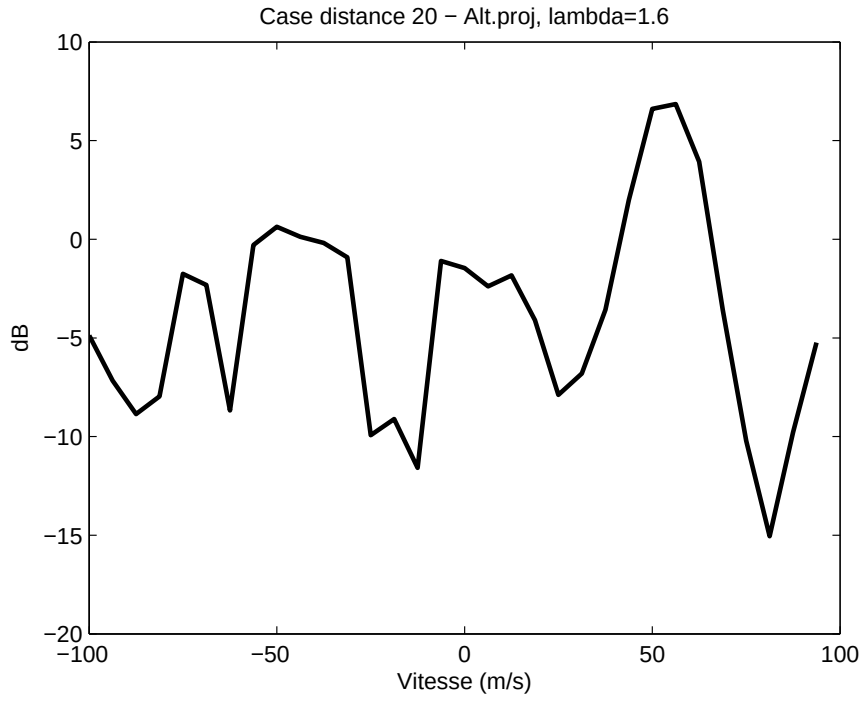


FIGURE 2.16 – Traitement STAP avec projection alternées, Lambda=1.6

2.2 Méthodes de sous-espace

2.2.1 Analyse de la méthode MLED

Pour étudier l'utilisation des méthodes de sous-espace, nous devons développer la formulation du détecteur MLED. En effet, ces méthodes ne peuvent fonctionner que si le sous-espace fouillis de la matrice d'origine $\mathbf{R}$ reste très proche du sous-espace fouillis de la matrice sans signal utile $\mathbf{Q}$. Pour une case distance donnée, si aucune cible n'est présente, la matrice de covariance $\mathbf{R}$ ne contient que des interférences, i.e fouillis et éventuellement des brouilleurs, et du bruit :

$$\mathbf{R} = \frac{\mathbf{X}\mathbf{X}^{\mathbf{H}}}{K_t} = \frac{\mathbf{N}\mathbf{N}^{\mathbf{H}}}{K_t} \tag{2.12}$$

La décomposition en éléments propres de $\mathbf{R}$, ou EVD pour *Eigen Value Decomposition*, nous permet de dissocier le sous-espace vectoriel des interférences du sous-espace bruit. Cependant, si une cible est présente à cette case distance, il n'est plus possible d'isoler correctement le sous-espace interférence parce que la cible fait partie du sous-espace dominant [1]. En effet, dans ce cas, la matrice de covariance $\mathbf{R}$ peut être écrite avec respectivement $\mathbf{s_{t(t)}}$ et $\mathbf{s_s}$ les *steering* vecteurs temporel et spatio-temporel de la cible :

$$\mathbf{R} = \frac{\mathbf{X}\mathbf{X}^{\mathbf{H}}}{K_t} = \frac{(\alpha \mathbf{s_s}\mathbf{s}_{\mathbf{t(t)}}^{\mathbf{T}} + \mathbf{N})(\alpha \mathbf{s_s}\mathbf{s}_{\mathbf{t(t)}}^{\mathbf{T}} + \mathbf{N})^{\mathbf{H}}}{K_t} \tag{2.13}$$

$$\mathbf{R} = \mid \alpha \mid^2 \mathbf{s_s}\mathbf{s}_{\mathbf{s}}^{\mathbf{H}} + \frac{\alpha \mathbf{s_s}\mathbf{s}_{\mathbf{t(t)}}^{\mathbf{T}}\mathbf{N}^{\mathbf{H}}}{K_t} + \frac{\alpha^* \mathbf{N}\mathbf{s}_{\mathbf{t(t)}}^{*}\mathbf{s}_{\mathbf{s}}^{\mathbf{H}}}{K_t} + \frac{\mathbf{N}\mathbf{N}^{\mathbf{H}}}{K_t} \tag{2.14}$$

Si α et $\mathbf{N}$ ne sont pas corrélés, et pour K_t grand, (2.14) peut être approximé par :

$$\mathbf{R} \approx \mid \alpha \mid^2 \mathbf{s_s}\mathbf{s}_{\mathbf{s}}^{\mathbf{H}} + \frac{\mathbf{N}\mathbf{N}^{\mathbf{H}}}{K_t} \neq \frac{\mathbf{N}\mathbf{N}^{\mathbf{H}}}{K_t} \tag{2.15}$$

Le résultat de l'Eq. (2.15) indique que la matrice de covariance $\mathbf{R}$ contient du signal interférence et du signal cible, si nous l'utilisons avec la méthode classique d'inversion matricielle (Annexe B.2), nous supprimerons les interférences du signal mais aussi les cibles. D'après (1.3) :

$$\mathbf{Q} = \mathbf{R} - \mathbf{g}\mathbf{g}^{\mathbf{H}}, \mathbf{g}\mathbf{g}^{\mathbf{H}} = \frac{\mathbf{X}\mathbf{s}_{\mathbf{t(D)}}^{*}\mathbf{s}_{\mathbf{t(D)}}^{\mathbf{T}}\mathbf{X}^{\mathbf{H}}}{K_t^2} \tag{2.16}$$

avec $\mathbf{s_{t(D)}}$ le *steering* vecteur temporel de la case Doppler sous test. En utilisant le modèle du signal cible, nous pouvons écrire :

$$\mathbf{g}\mathbf{g}^{\mathbf{H}} = \frac{(\alpha \mathbf{s_s}\mathbf{s}_{\mathbf{t(t)}}^{\mathbf{T}} + \mathbf{N})\mathbf{s}_{\mathbf{t(D)}}^{*}\mathbf{s}_{\mathbf{t(D)}}^{\mathbf{T}}(\alpha \mathbf{s_s}\mathbf{s_{t(t)}}^{\mathbf{T}} + \mathbf{N})^{\mathbf{H}}}{K_t^2} \tag{2.17}$$

1. Le sous-espace dominant est engendré par les vecteurs propres associés aux valeurs propres supérieures à la variance du bruit

En utilisant la même approximation que dans (2.15), (2.17) mène à :

$$\mathbf{gg^H} \approx \frac{\mid \alpha \mid^2 \mathbf{s_s s^T_{t(t)} s^*_{t(D)} s^T_{t(D)} s^*_{t(t)} s^H_s} + \mathbf{N s^*_{t(D)} s^T_{t(D)} N^H}}{K_t^2} \tag{2.18}$$

Notons

$$\rho = \mathbf{s^T_{t(t)} s^*_{t(D)} s^T_{t(D)} s^*_{t(t)}} =\mid \mathbf{s^T_{t(t)} s^*_{t(D)}} \mid^{\mathbf{2}} \tag{2.19}$$

La matrice de covariance modifiée $\mathbf{Q}$ devient alors :

$$\mathbf{Q} \approx (1 - \frac{\rho}{K_t^2}) \mid \alpha \mid^2 \mathbf{s_s s^H_s} + \frac{\mathbf{NN^H}}{K_t} - \frac{\mathbf{N s^*_{t(D)} s^T_{t(D)} N^H}}{K_t^2} \tag{2.20}$$

Lorsque l'on teste la case Doppler de la cible, i.e $\mathbf{s_{t(D)}} = \mathbf{s_{t(t)}} = \mathbf{s_t}$, $\rho = K_t^2$, (2.20) devient :

$$\mathbf{gg}^H = \frac{(\alpha \mathbf{s_s s^T_t s^*_t} + \mathbf{N s^*_t})(\alpha^* \mathbf{s^T_t s^*_t s^H_s} + \mathbf{s^T_t N^H})}{K_t^2} \tag{2.21}$$

et

$$\mathbf{gg^H} =\mid \alpha \mid^2 \mathbf{s_s s^H_s} + \frac{\alpha \mathbf{s_s s^T_t N^H}}{K_t} + \frac{\alpha^* \mathbf{N s^*_t s^H_s}}{K_t} + \frac{\mathbf{N s^*_t s^T_t N^H}}{K_t^2} \tag{2.22}$$

Donc de (2.14), (1.3) et (2.22), la matrice $\mathbf{Q}$ est, sans approximation :

$$\mathbf{Q} = \frac{\mathbf{NN^H}}{K_t} - \frac{\mathbf{N s^*_t s^T_t N^H}}{K_t^2} \tag{2.23}$$

La matrice $(\mathbf{NN^H})/K_t$ est la matrice de covariance estimée des interférences et du bruit alors que $(\mathbf{N s^*_t s^T_t N^H})/K_t^2$ est la matrice de covariance des vecteurs interférence plus bruit projetés sur $\mathbf{s^*_t}$. Il vient de (2.23) que la matrice de covariance modifiée $\mathbf{Q}$ utilisée pour le détecteur MLED dans (1.5) ne contient plus de signal d'intérêt et que le signal cible ne sera plus supprimé par le filtre STAP (1.2) contrairement aux interférences. Notons que la matrice résiduelle contenant les interférences et le bruit est légèrement différente de la vraie matrice de covariance estimée $(\mathbf{NN^H})/K_t$.
Si la case Doppler testée est différente de la case Doppler de la cible $\mathbf{s_{t(D)}} \neq \mathbf{s_{t(t)}}$, et comme ρ est essentiellement un $sinc^2$, il décroit très rapidement, $\rho \longrightarrow 0$. Si nous faisons la même approximation que dans (2.15) et (2.20), nous avons

$$\mathbf{Q} \approx\mid \alpha \mid^2 \mathbf{s_s s^H_s} + \frac{\mathbf{NN^H}}{K_t} - \frac{\mathbf{N s^*_{t(D)} s^T_{t(D)} N^H}}{K_t^2} \tag{2.24}$$

Lorsque $\rho \neq K_t^2$, le signal cible est toujours dans la matrice de covariance $\mathbf{Q}$, comme dans $\mathbf{R}$, mais cela n'a pas d'effet sur le filtre parce que le *steering* vecteur spatio-temporel de la case Doppler $\mathbf{s_{s(D)}}$ est différent [2] du *steering* vecteur de la cible $\mathbf{s_{s(t)}}$.

2. rappelons que ce *steering* vecteur spatio-temporel $\mathbf{s_s}$ est en fait $\mathbf{s_s} = \mathbf{s_1} \otimes \mathbf{s_2}$ où $\mathbf{s_1}$ et $\mathbf{s_2}$ sont des *steering* vecteurs purement spatiaux et purement temporel, respectivement, et où $\otimes$ est le produit de Kronecker

En effet, introduisons la décomposition en éléments propres de $\mathbf{Q}$ quand $\rho = K_t^2$ dans (2.23) par :

$$\mathbf{Q} = \mathbf{V_N}\mathbf{\Lambda_N}\mathbf{V_N^H} \tag{2.25}$$

En supposant que $\mathbf{s_{s(t)}}$ est orthogonal au sous-espace engendré par les interférences et le bruit $\mathbf{V_N}$, nous pouvons écrire la décomposition en éléments propres de $\mathbf{Q}$ lorsque $\rho \neq K_t^2$ ($\mathbf{s_{t(t)}} \neq \mathbf{s_{t(D)}}$) comme :

$$\mathbf{Q} = \lambda_1'\mathbf{V_1'}\mathbf{V_1'^H} + \mathbf{V_N'}\mathbf{\Lambda_N'}\mathbf{V_N'^H} \tag{2.26}$$

avec $\mathbf{s_{s(t)}} = \gamma\mathbf{V_1'}$. De la même manière, nous pouvons écrire l'EVD de $\mathbf{R}$ dans (2.15) :

$$\mathbf{R} = \lambda_1''\mathbf{V_1''}\mathbf{V_1''^H} + \mathbf{V_N''}\mathbf{\Lambda_N''}\mathbf{V_N''^H} \tag{2.27}$$

Il en découle que le terme $\mathbf{s_{s(D)}^H Q^{-1}}$ apparaissant dans l'équation (1.2) du Chapitre 1, lorsque $\rho \neq K_t^2$ est

$$\mathbf{s_{s(D)}^H Q^{-1}} = \frac{1}{\lambda_1'}\mathbf{s_{s(D)}^H}\mathbf{V_1'}\mathbf{V_1'^H} + \mathbf{s_{s(D)}^H}\mathbf{V_N'}\mathbf{\Lambda_N'^{-1}}\mathbf{V_N'^H} \tag{2.28}$$

et peut être écrit

$$\mathbf{s_{s(D)}^H Q^{-1}} = \frac{1}{\lambda_1'\gamma^2}\mathbf{s_{s(D)}^H s_{s(t)} s_{s(t)}^H} + \mathbf{s_{s(D)}^H}\mathbf{V_N'}\mathbf{\Lambda_N'^{-1}}\mathbf{V_N'^H} \tag{2.29}$$

Notons que de la même façon, le terme $\mathbf{s_{s(D)}^H R^{-1}}$ mène à

$$\mathbf{s_{s(D)}^H R^{-1}} = \frac{1}{\lambda''_{\ 1}\gamma^2}\mathbf{s_{s(D)}^H s_{s(t)} s_{s(t)}^H} + \mathbf{s_{s(D)}}^{\mathbf{H}}\mathbf{V_N''}\mathbf{\Lambda_N''^{-1}}\mathbf{V_N''^H} \tag{2.30}$$

Si $\mathbf{s_{s(D)}}$ et $\mathbf{s_{s(t)}}$ ne sont pas proches, le terme $\mathbf{s_{s(D)}^H s_{s(t)}}$ est petit et la présence de la cible dans les matrices de covariance $\mathbf{Q}$ et $\mathbf{R}$ n'a pas d'effet sur les filtres. Lorsque $\mathbf{s_{s(D)}}$ et $\mathbf{s_{s(t)}}$ deviennent proches, $\mathbf{Q}$ se réduit à (2.23) et a donc la décomposition (2.25), ainsi $\mathbf{s_{s(D)}^H Q^{-1}}$ devient

$$\mathbf{s_{s(D)}^H Q^{-1}} = \mathbf{s_{s(D)}^H}\mathbf{V_N'}\mathbf{\Lambda_N'^{-1}}\mathbf{V_N'^H} \tag{2.31}$$

La cible n'a toujours pas d'effet, alors que le premier terme de la partie droite de (2.29) qui contient la contribution de la cible augmente. Nous nous attendons donc à observer une perte du rapport signal à bruit SINR autour de la case Doppler de la cible, excepté à la case Doppler de la cible à cause de la finesse du projecteur MLED (voir Section 1.1).

2.2.2 EC-APES

La projection d'Hung-Turner, aussi appelée *Eigencanceller* (EC) est basée sur une décomposition en éléments propres de la matrice de covariance [15]. Cette technique est bien plus robuste face à la mauvaise estimation de la matrice de covariance que ne l'est l'inversion matricielle [47]. Le filtre STAP avec *eigencanceller* est :

$$\mathbf{w} = (\mathbf{I} - \mathbf{V_c}\mathbf{V_c^H})\mathbf{s_s} \tag{2.32}$$

avec $\mathbf{I}$ la matrice identité de dimension NM et où $\mathbf{V_c}$ est la matrice de dimension $MN \times p$ contenant les vecteurs propres de $\mathbf{R}$ orthonormés associés aux p valeurs propres strictement supérieures à la variance du bruit (le bruit est supposé Gaussien, i.i.d. et de variance σ^2). Le détecteur s'écrit alors :

$$\textit{Détecteur EC :}\ \frac{|\mathbf{s_s^H}(\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{X}|^2}{\mathbf{s_s^H}(\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{2.33}$$

Lorsqu'il n'y a aucune cible ni aucun brouilleur dans les signaux, p est le rang de la matrice de covariance fouillis seul. D'après la loi de Brennan [18] :

$$p = M + \beta(N-1) \tag{2.34}$$

dans le cas d'une antenne ALU [3]. Dans le cas où il y a J brouilleurs et T cibles, il y a $p' = p + J + T$ vecteurs propres supérieurs à la variance du bruit. Nous supposerons dans la suite qu'il n'y a pas de brouilleurs et qu'une seule cible est présente dans une case Doppler donnée. Le filtre EC-APES est déduit de l'EVD de la matrice $\mathbf{Q}$:

$$\mathbf{Q} = \mathbf{V_Q}\mathbf{\Lambda_Q}\mathbf{V_Q^H} \tag{2.35}$$

Il s'écrit alors

$$\textit{Détecteur EC-APES :}\ \frac{|\mathbf{s_s^H}(\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{g}|^2}{\mathbf{s_s^H}(\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{2.36}$$

avec $\mathbf{V_c}$ qui sont cette fois les vecteurs propres de la matrice $\mathbf{Q}$ (et $\mathbf{V_b}$ du sous-espace bruit. Notons :

$$\mathbf{X} = \alpha\mathbf{s_s} + \mathbf{C} + \mathbf{N'} \tag{2.37}$$

le signal à la case Doppler de la cible. Il est composé du signal cible, d'une composante de fouillis $\mathbf{C}$ et de bruit $\mathbf{N'}$. Le projecteur $(\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})$ apparaissant dans (2.32) et appliqué à $\mathbf{X}$ donne :

$$(\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{X} = (\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\alpha\mathbf{s_s} + (\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{C} + (\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{N'} \tag{2.38}$$

d'où

$$(\mathbf{I}-\mathbf{V_c}\mathbf{V_c^H})\mathbf{X} = \alpha\mathbf{s_s} + \mathbf{N'} \tag{2.39}$$

Le fouillis a été supprimé alors que le premier terme de la partie droite de l'équation (2.38) n'a pas disparu parce que $\alpha\mathbf{s_s}$ n'est pas dans le sous-espace dominant fouillis seul.
Au contraire, comme la cible est présente dans $\mathbf{R}$, notons :

$$\mathbf{R} = \mathbf{V_{c+t}}\mathbf{\Lambda_{c+t}}\mathbf{V_{c+t}^H} + \mathbf{V_b}\mathbf{\Lambda_b}\mathbf{V_b^H} \tag{2.40}$$

où $\mathbf{V_{c+t}}$ contient les $p+1$ vecteurs propres dominants de $\mathbf{R}$ engendrant le sous-espace fouillis plus cible et où $\mathbf{V_b}$ contient les $MN-p-1$ vecteurs propres engendrant le sous-espace bruit seul. Nous pouvons facilement en déduire que :

$$(\mathbf{I}-\mathbf{V_{c+t}}\mathbf{V_{c+t}^H})\mathbf{X} = (\mathbf{I}-\mathbf{V_{c+t}}\mathbf{V_{c+t}^H})\mathbf{N'} \tag{2.41}$$

comme la cible $\alpha\mathbf{s_s}$ fait partie du sous-espace fouillis+cible et est orthogonal au sous-espace bruit.

3. $\beta = \frac{2V_a}{\lambda}T_r$ où V_a est la vitesse du porteur, λ est la longueur d'onde et T_r est l'intervalle de répétition des impulsions ou PRI

En conséquence, en sortie du filtre STAP EC, la cible a été supprimée et sa détection n'est plus possible contrairement à EC-APES.
Cette méthode de sous-espace ne requiert que $2p$ échantillons de données pour converger à une perte de signal utile de $-3dB$ comparés aux $2MN$ échantillons nécessaires avec la méthode d'inversion matricielle SMI. Ce détecteur a par contre un coût calculatoire plus élevé que le MLED ou le Stop-Band à cause de l'EVD de la matrice $\mathbf{Q}$ qui doit être effectuée pour toutes les cases Doppler et pour toutes les cases distance.

2.2.3 FAPI-APES

Dans cette section, nous nous intéressons à l'algorithme *Fast Approximated Power Iteration* (FAPI) [48] appliqué à APES. L'algorithme FAPI construit une base de vecteurs orthonormaux $\mathbf{W_c}$ qui engendrent le sous-espace formé par les p vecteurs propres dominants de la décomposition en élements propres de la matrice de covariance estimée avec un ensemble de K_t vecteurs de données $\mathbf{x}_k$ qui sont les vecteurs spatio-temporels qui forment les colonnes de la matrice de données $\mathbf{X}$. Nous supposerons ici que nous connaissons p grâce à la formule de Brennan, voir Eq. (2.34). L'algorithme FAPI est le suivant :

Initialisation : $\mathbf{W_c}(0) \leftarrow \mathbf{I}_{MN*r}, \mathbf{Z}(0) \leftarrow \mathbf{I}_{p*p}$
for $k = 1$ to K_t
$\mathbf{y}(k) = \mathbf{W_c}(k-1)^H \mathbf{x}_k$
$\mathbf{h}(k) = \mathbf{Z}(k-1).\mathbf{y}(\mathbf{k})$
$\mathbf{g}(k) = \frac{\mathbf{h(k)}}{\mathbf{y}^H(k)\mathbf{h(k)}+\beta}$
$\mathbf{e}(k) = \mathbf{x}_k - \mathbf{W_c}(k-1)\mathbf{y}(k)$
$\epsilon^2(k) = \|\mathbf{x}_k^2\| - \|\mathbf{y}_k^2\|$
$\tau(k) = \frac{\epsilon^2}{1+\epsilon^2\|\mathbf{g}(k)\|^2+\sqrt{1+\epsilon^2\|\mathbf{g}(k)\|^2}}$
$\eta(k) = 1 - \tau(k)\|\mathbf{g}(k)\|^2$
$\mathbf{y}'(k) = \eta(k)\mathbf{y}(k) + \tau(k)\mathbf{g}(k)$
$\mathbf{h}'(k) = \mathbf{Z}(k-1)^H \mathbf{y}'(k)$
$\mathbf{d}(k) = \frac{\tau(k)}{\eta(k)}(\mathbf{Z}(k-1)\mathbf{g}(k) - (\mathbf{h}'(k)\mathbf{g}(k))\mathbf{g}(k))$
$\mathbf{Z}(k) = \frac{1}{\beta}(\mathbf{Z}(k-1) - \mathbf{g}(k)\mathbf{h}'(k)^H + \mathbf{d}(k)\mathbf{g}(k)^H)$
$\mathbf{e}'(k) = \eta(k)\mathbf{x}_k - \mathbf{W_c}(k-1)y'(k)$
$\mathbf{W}(k) = \mathbf{W}(k-1) + \mathbf{e}'(k).\mathbf{g}(k)^H$
ENDFOR

Algorithme FAPI

La matrice résultante $\mathbf{W_c}$ contient les p vecteurs dominants qui engendrent la même base que les p vecteurs propres dominants de $\mathbf{R} = \frac{\mathbf{1}}{\mathbf{K_t}} \sum_{\mathbf{k=1}}^{\mathbf{K_t}} \mathbf{x_k x_k^H}$

Dans [49], une méthode récursive STAP reposant sur FAPI a été proposée. Dans ce cas, les vecteurs de données $\mathbf{x}_k$ sont pris à d'autres cases distance, constituant ainsi les données secondaires comme pour les traitements STAP traditionnels. Ici, les vecteurs de données sont pris sur la dimension temps long, c'est à dire sur les récurrences radar de la case distance sous test et aucune matrice de covariance n'est estimée.

Pour pouvoir appliquer un traitement analogue à EC-APES (voir Section 2.2.2), nous devons retirer le signal d'intérêt directement des vecteurs de données de la façon suivante :

$$\mathbf{Y} = \mathbf{XU} \ / \ \mathbf{YY^H} = \mathbf{Q} \tag{2.42}$$

avec

$$\mathbf{U} = \mathbf{I}_{K_\mathbf{t}} - \frac{\mathbf{s_t^* s_t^T}}{K_\mathbf{t}} \tag{2.43}$$

Le vecteur $\mathbf{y_k}$ qui est la k^{me} colonne de $\mathbf{Y}$ est utilisé dans l'algorithme FAPI à la place de $\mathbf{x_k}$ pour calculer la base de vecteurs $\mathbf{W}_c$. Une fois que $\mathbf{W}_c$ est calculé, les poids du filtre peuvent être écrits comme dans l'équation (2.32) :

$$\mathbf{w} = (\mathbf{I} - \mathbf{W_c W_c^H})\mathbf{s_s} \tag{2.44}$$

La détection est alors déduite de l'équation (2.33) :

$$\textit{Détecteur FAPI-APES} : \frac{|\mathbf{s_s^H}(\mathbf{I} - \mathbf{W_c W_c^H})\mathbf{g}|^\mathbf{2}}{\mathbf{s_s^H}(\mathbf{I} - \mathbf{W_c W_c^H})\mathbf{s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \tag{2.45}$$

La charge calculatoire de cette méthode est plus faible que celle de EC-APES ou que du MLED standard parce que le coût calculatoire de cet algorithme est en $o(MNp)$ alors qu'il est en $o((MN)^3)$ pour les méthodes MLED ou EC-APES. Les charges calculatoires des différents algorithmes sont discutées dans le Chapitre 1 de la Partie III.

2.2.4 Détermination du sous-espace interférence

Dans le chapitre précédant, nous avons utilisé la loi de Brennan pour obtenir une approximation du rang de la matrice et ainsi isoler le sous-espace interférences (fouillis et brouilleurs) du sous-espace bruit et éventuellement cible. Cependant, dans une application pratique, ou dans une situation différente du cas d'une antenne linéaire uniforme à visée latérale, il peut être nécessaire d'avoir une indication plus correcte du rang de la matrice de covariance. Des méthodes de détermination du nombre de sources basées sur la décroissance des valeurs propres de la matrice de covariance sont bien documentées dans la littérature. Ces critères cherchent une discontinuité dans la décroissance des valeurs propres pour déterminer la limite entre les différents sous-espaces. Nous proposerons deux nouvelles méthodes pour déterminer le rang de la matrice.

Détermination du nombre de sources à l'aide de critères sur les valeurs propres

Les critères AIC et MDL [50][51] sont deux critères utilisés pour la détermination du sous-espace interférence à partir du spectre de valeurs propres $\{\lambda_i\}_{i=1,\ldots,NM}$ de la matrice de covariance.

Critère d'Akaike :

$$AIC(p) = -K_t \log \frac{\prod_{i=p+1}^{MN} \lambda_i}{(\frac{1}{MN-p}\sum_{i=p+1}^{MN} \lambda_i)^{MN-p}} + p(2MN - p) \tag{2.46}$$

Critère MDL :

$$MDL(p) = -K_t \log \frac{\prod_{i=p+1}^{MN} \lambda_i}{(\frac{1}{MN-p} \sum_{i=p+1}^{MN} \lambda_i)^{MN-p}} + \frac{1}{2} P(2MN - p) \log K_t \qquad (2.47)$$

Le nombre de sources p_{min} est déterminé par les valeurs minimales de $AIC(p)$ ou $MDL(p)$. Nous testons ces critères sur les données **SimALU** avec les paramètres suivants :

Type	Case distance	Nombre voies (N)	Ktaps (M)	Nombre estimées (K_t)
SimALU	100	4	6	59

Sur la Figure 2.17, la frontière entre sous-espace fouillis et sous-espace bruit est clairement visible, elle se situe à la valeur propre numéro 7. Les critères MDL et AIC affichés sur les Figures 2.18 et 2.19 indiquent tous les deux un sous-espace fouillis de dimension 7.

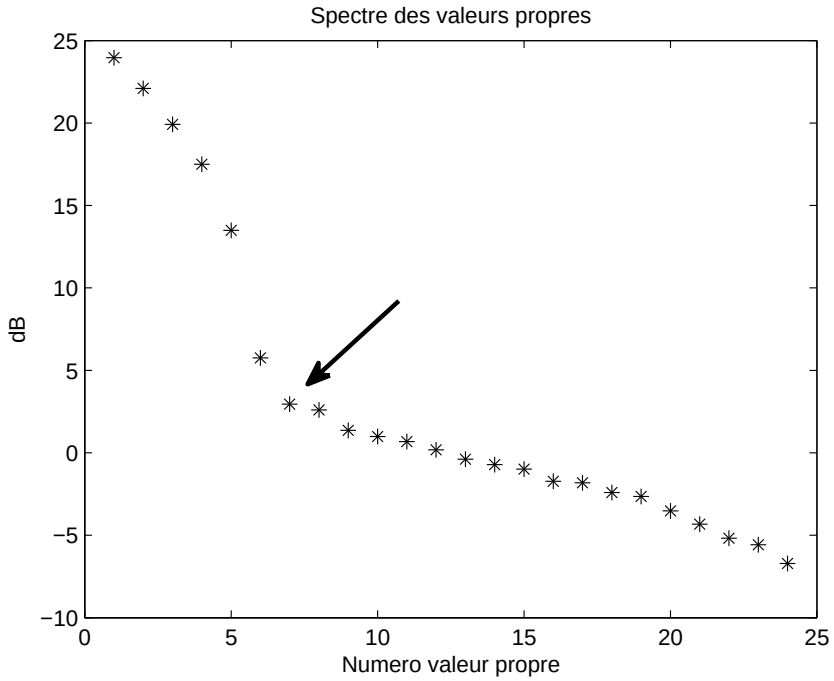


FIGURE 2.17 – Spectre des valeurs propres de la matrice de covariance - Données SimALU

Rappelons que la loi de Brennan donne une dimension de $p = M + \beta(N-1) = 9$. Pour comparer ces deux dimensions, nous utilisons le rapport interférence à bruit appelé aussi *Clutter to Noise Ratio* ou CNR (voir Annexe B.5.1). Pour une dimension de fouillis égale à 7, le rapport interférence à bruit est *CNR=-0.07 dB* alors qu'il est égal à *CNR=-0.13 dB* pour une dimension de 9, ces deux valeurs sont acceptables pour un niveau de réjection de fouillis, celui-ci tend vers 0 avec un filtre optimal. Le rang de Brennan surestime légèrement la dimension du sous-espace fouillis. Cette surestimation peut causer une dégradation du SNR d'une éventuelle cible qui se trouvera dans ce sous-espace surestimé. En revanche, les deux critères précédemment décris ont correctement estimé la dimension du sous-espace fouillis.

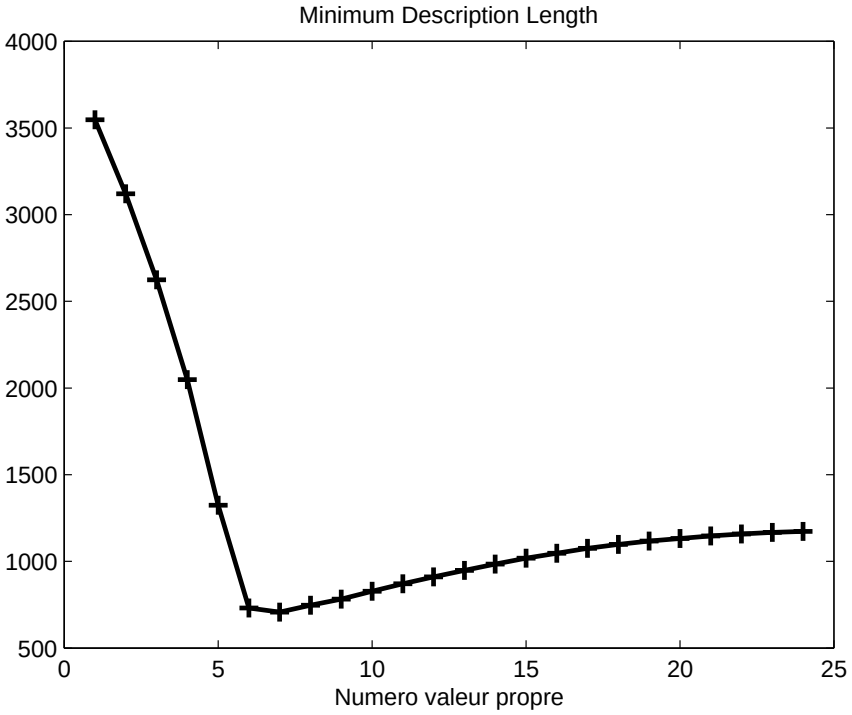


FIGURE 2.18 – Critère MDL sur les valeurs propres de la matrice de covariance

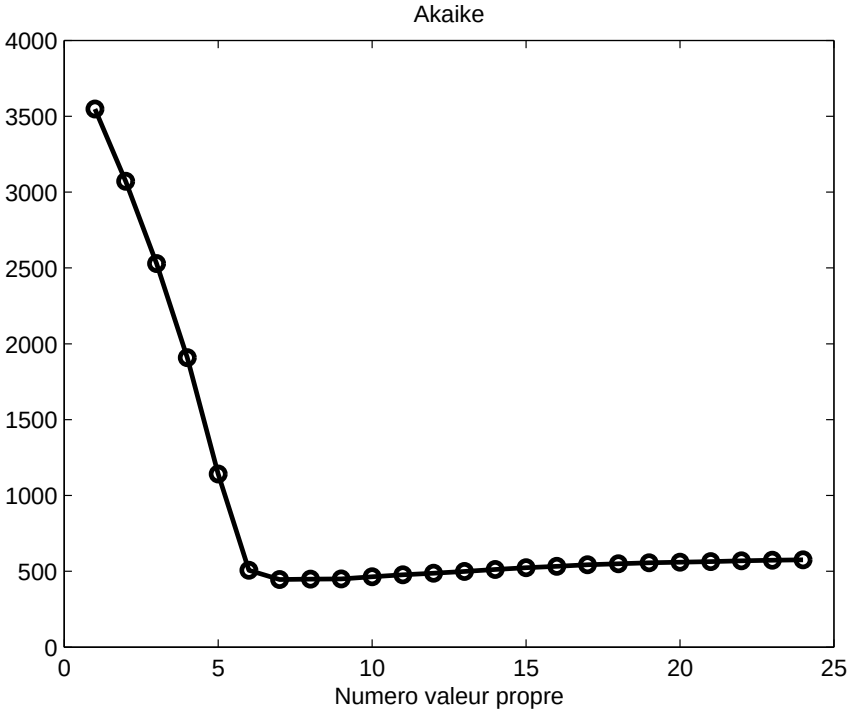


FIGURE 2.19 – Critère AIC sur les valeurs propres de la matrice de covariance

La géométrie et le type d'antenne, les erreurs de phases et le type d'interférences agissent sur les valeurs propres de la matrice de covariance. Dans certaines situations il n'y a pas de changement brutal dans le spectre de valeurs propres pouvant rendre ces critères moins efficaces. Nous testons maintenant ces deux critères sur des données synthétiques réalistes **ONERA-AS**. L'antenne utilisée étant pondérée, il faut blanchir la matrice de covariance avant d'en faire l'EVD de manière à ne pas biaiser les valeurs propres (voir Annexe B.5.3). Les paramètres du test sont les suivants :

Type	Case distance	Nombre voies (N)	Ktaps (M)	Nombre estimées (K_t)
ONERA AS	750	8	7	58

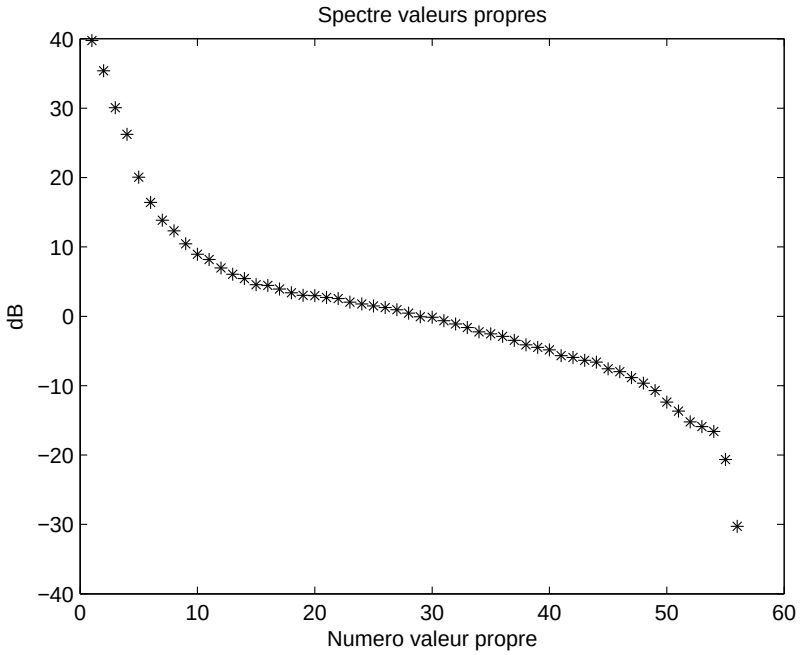


FIGURE 2.20 – Spectre valeurs propres de la matrice de covariance

La Figure 2.20 montre qu'il n'est pas possible de distinguer de frontière entre valeurs propres associées au fouillis et valeurs propres associées au bruit. Les dimensions estimés de fouillis calculés à l'aide des deux critères, du rang de Brennan sont comparées dans le tableau suivant :

Méthode	Brennan	Critère AIC	Critère MDL
Dimension fouillis	21	56	12
CNR Ratio	-0.44 dB	-30 dB	-0.12 dB

Le critère MDL permet d'obtenir un niveau de réjection correct avec une dimension de fouillis minimum. Le critère AIC n'arrive pas à déterminer une dimension correcte et retourne le nombre total de valeurs propres. Le rang de Brennan quant à lui surestime toujours le sous-espace fouillis.

Dans la partie suivante, nous proposerons deux nouvelles méthodes visant à déterminer correctement la dimension du sous-espace interférence.

Théorie des matrices aléatoires : bornes sur les valeurs propres

Comme nous l'avons vu sur la Figure 2.20, il peut y avoir une continuité entre les valeurs propres associées au fouillis et celles associées au bruit. En effet, lorsque l'on estime une matrice de covariance à partir d'un jeux de vecteurs de bruit Gaussien, le spectre des valeurs propres est perturbé par l'estimation et les valeurs propres correspondant au bruit peuvent être très proches des valeurs propres associées au fouillis. Dans le cas de vecteurs de donnés $\mathbf{x}_m$ contenant uniquement du bruit Gaussien de variance unité, la vraie matrice de covariance est égale à la matrice identité.

$$\mathbf{x}_m \mathbf{x^H}_m = \mathbf{I} \tag{2.48}$$

Les valeurs propres associées aux vecteurs propres de la matrice identité étant égales à 1, le spectre des valeurs propres d'une matrice de covariance de bruit estimée sur une infinité d'échantillons tend vers 1. Le nombre d'échantillons étant toujours fini, et souvent insuffisant dans notre cas, les valeurs propres associés au bruit s'étendront d'une valeur maximum $\lambda_b max > 1$ à une valeur minimum $\lambda_b min < 1$. La valeur propre minimum associée au sous-espace interférence peut être très proche de la valeur propre maximum associé au sous-espace bruit. Cela implique une relative continuité dans la décroissance des valeurs propres de la matrice de covariance, pouvant rendre les algorithmes étudiés précédemment moins performants.
Sur la Figure 2.21 sont affichées les valeurs propres des matrices de covariance calculées avec des données ne contenant que du bruit, avec les mêmes paramètres que pour l'essai précédent : 8 voies, 7 Ktaps et un nombre d'échantillons d'estimation K_t de 58, 116 et 580. Les spectres de valeurs propres de la Figure 2.21 montrent bien le biais induit par le nombre d'estimées, plus il est faible, plus $\lambda_b min$ et surtout $\lambda_b max$ s'éloignent de leur vraie valeur, et risquant ainsi de se confondre avec les valeurs propres du sous-espace fouillis dans le cas de données avec fouillis.
Nous allons utiliser cet effet de déformation du spectre des valeurs propres associées au sous-espace bruit pour borner ce sous-espace et ainsi déterminer une limite entre valeurs propres associées au sous-espace interférence et au sous-espace bruit. Dans [52] et [53], les auteurs démontrent que pour un vecteur de données aléatoire de taille M, sous certaines conditions comme $\mathbf{x_m} \sim \mathcal{CN}(1, \mathbf{I})$, la distribution des valeurs propres converge vers une fonction non-aléatoire $f(x)$ et que la valeur propre maximale est biaisée vers le haut et converge vers une valeur c_S alors que la valeur propre minimale est biaisée vers le bas et converge vers une valeur c_I.
Soit K_t échantillons $\mathbf{x} \sim \mathcal{CN}(1, \mathbf{I})$ de taille MN et soit $a = \frac{K_t}{MN}$ le rapport entre le nombre d'échantillons et la taille du vecteur. Les valeurs c_S et c_I sont définies par :

$$c_S = \frac{1}{a}(\sqrt{a} + 1)^2 \tag{2.49}$$

$$c_I = \frac{1}{a}(\sqrt{a} - 1)^2 \tag{2.50}$$

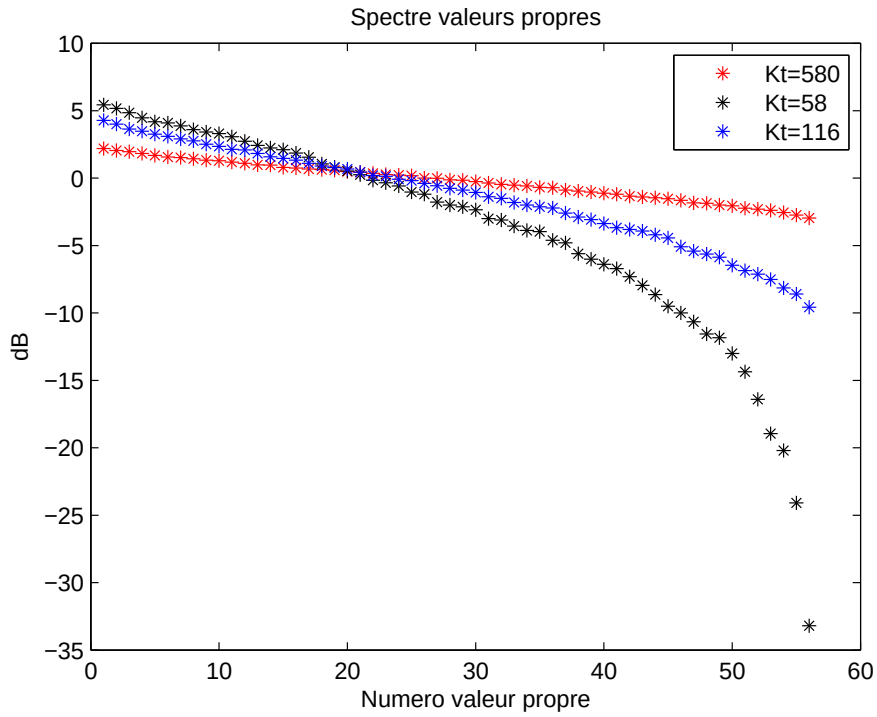


FIGURE 2.21 – Spectre valeurs propres - bruit seul

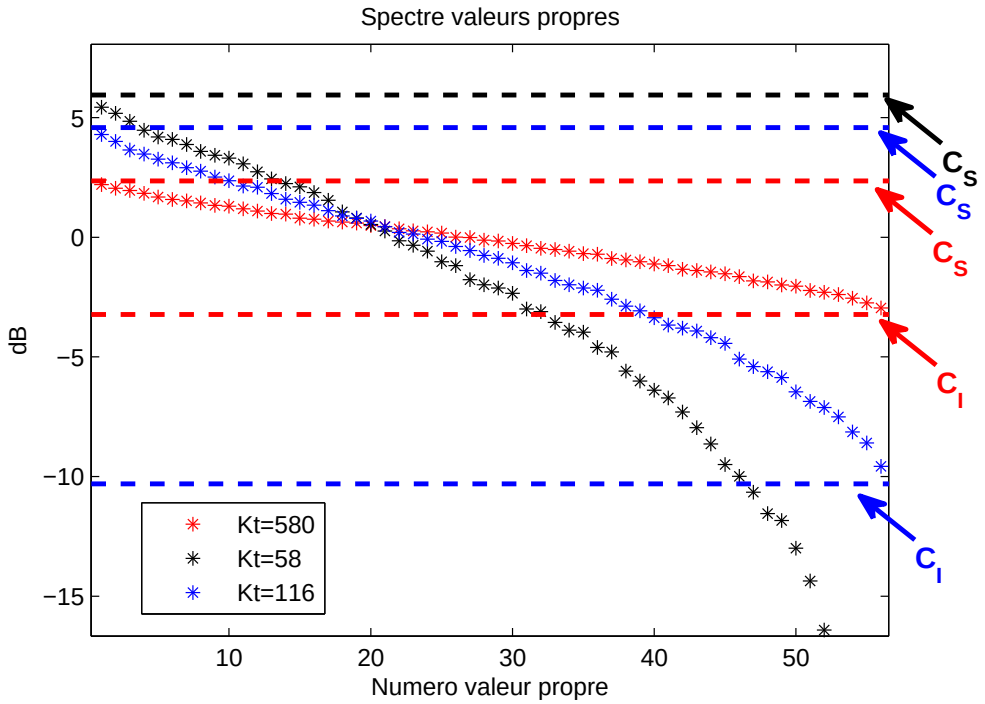


FIGURE 2.22 – Spectre valeurs propres et bornes - bruit seul

Ces bornes sont testées sur les valeurs propres affichées sur la Figure 2.22.
Les résultats de la Figure 2.22 montrent que c_S et c_I bornent très bien les différentes distributions de valeurs propres obtenues avec des nombres d'échantillons différents. Pour déterminer la dimension du sous-espace interférences, notre méthode utilisera la valeur c_S comme valeur maximale de la dimension du sous-espace bruit.

Méthode utilisant le CNR et performances

De la même manière que nous utilisons le CNR pour comparer les différentes méthodes, nous pouvons utiliser ce CNR en calculant sa valeur pour chaque dimension du sous-espace fouillis. Le CNR optimal visé étant de 0 dB, un seuil arbitraire légèrement supérieur ou inférieur peut être fixé. L'avantage de cette méthode est le fait qu'elle ne requiert pas une connaissance des valeurs propres mais seulement des vecteurs propres. Nous utilisons toujours les mêmes paramètres sur les données **ONERA-AS** et nous affichons la valeur du CNR en fonction de la dimension du sous-espace fouillis choisie.

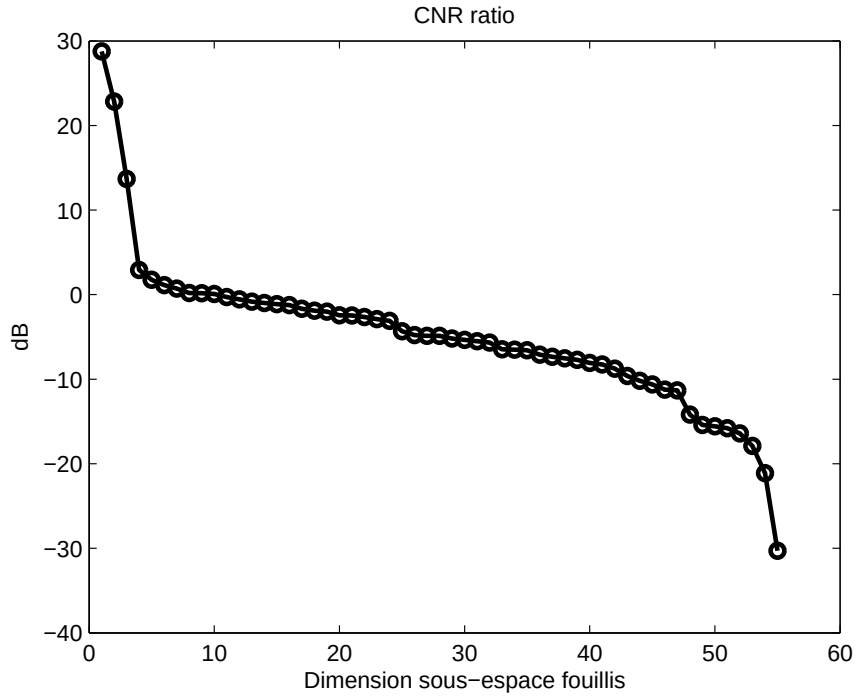


FIGURE 2.23 – CNR Ratio +1 en fonction de la dimension du sous-espace fouillis

Nous fixons un seuil de CNR à 0dB pour la méthode basée sur le CNR, et nous comparons la dimension trouvée avec les méthodes précédemment décrites :

Méthode	Brennan	Critère AIC	Critère MDL	Bornes val. pro.	Seuil CNR
Dimension fouillis	21	56	12	14	11
CNR Ratio	-0.44 dB	-30 dB	-0.12 dB	-0.18 dB	-0.05 dB

Comme nous pouvons le voir les méthodes basées sur le critère MDL, sur les bornes des valeurs propres bruit et sur le seuil du CNR affichent des résultats très proches, bien que la méthode sur les bornes soit la plus robuste puisqu'elle ne dépend pas des valeurs propres du fouillis. La méthode du CNR (commes les critères MDL et AIC) demande le plus de calculs alors que la méthode utilisant les bornes est, avec la loi de Brennan, celle qui en demande le moins. Dans la suite, nous choisissons d'utiliser la méthode des bornes lors de l'utilisation de l'algorithme EC-APES et la méthode du CNR avec l'algorithme FAPI-APES. Cette méthode est la plus robuste pour faire face aux situations fortement hétérogènes.

2.2.5 Contrôle des lobes secondaires

Le principal objectif du filtrage adaptatif est de supprimer les signaux interférence tout en gardant le diagramme d'antenne dans la direction désirée. Cependant, la méthode d'inversion matricielle peut engendrer un diagramme d'antenne avec des lobes secondaires importants dans des directions éloignées de celles d'où viennent les interférences. Ceci est un problème important pour la localisation des cibles. L'algorithme SMI minimise la puissance moyenne en sortie avec une contrainte, généralement $\mathbf{w^H s_s} = \mathbf{1}$, les poids résultants étant ceux utilisés dans le traitement STAP classique :

$$\mathbf{w_{SMI}^H} = \frac{\mathbf{s_s^H R^{-1}}}{\mathbf{s_s^H R^{-1} s_s}} \tag{2.51}$$

L'algorithme SMI (ainsi que le détecteur MLED qui est basé sur ce dernier) avec une seule contrainte sur le gain dans une direction donnée n'a pas de mécanisme qui génère des lobes secondaires faibles, et si nous n'utilisons pas un grand nombre de données pour l'estimation de $\mathbf{R}$, des corrélations parasites entre termes croisés peuvent conduire à une hausse importante des lobes secondaires comme nous pouvons le constater sur la courbes en pointillée de la Figure 2.24. Pour résoudre cet effet indésirable, une méthode est proposée dans [54], appelée *Constrained Signal Subspace Projection* (CSSP) qui est basée sur une méthode proposée dans [55]. Le vecteur des poids adaptatifs obtenu en utilisant la méthode SMI est projeté dans le sous-espace interférence plus une contrainte $\mathbf{s_s}$. Le sous-espace interférence $\mathbf{J}$ est estimé de la même façon que dans la Section précédente pour EC-APES (Section 2.2.2).

$$\mathbf{w_{CSSP}} = \mathbf{P_{[s,J]} w_{SMI}} \tag{2.52}$$

où $\mathbf{P_{[s,J]}}$ est l'opérateur de projection dans le sous-espace engendré par la base de vecteur interférence et par le vecteur contrainte.

Les méthodes EC-APES et FAPI-APES par contre, ne souffrent pas de ce phénomène puisque les poids du filtre sont directement le résultat de la projection du *steering* vecteur $\mathbf{s_s}$ dans le sous-espace qui est orthogonal au sous-espace interférence comme dessiné sur la Figure 2.25. Sur la Figure 2.24, la différence entre MLED et FAPI-APES est clairement visible sur les lobes secondaires. Le CSSP appliqué sur le MLED et sur FAPI-APES n'est pas affiché sur la figure parce que ses performances sont identiques à celles de FAPI-APES.

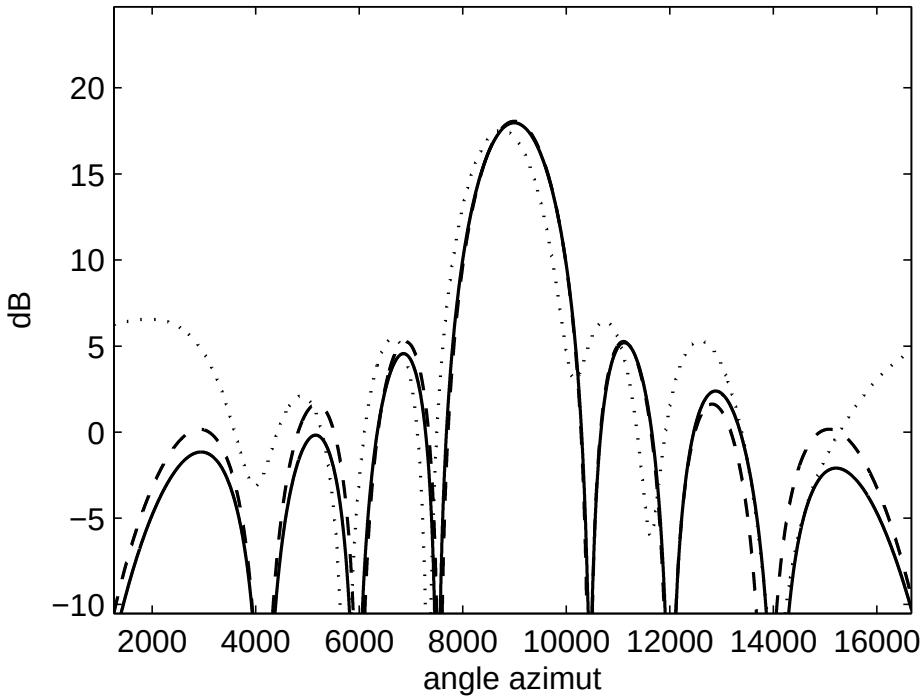


FIGURE 2.24 – Poids au repos (courbe pleine), MLED (courbe pointillée) et FAPI-APES (courbe tirets)

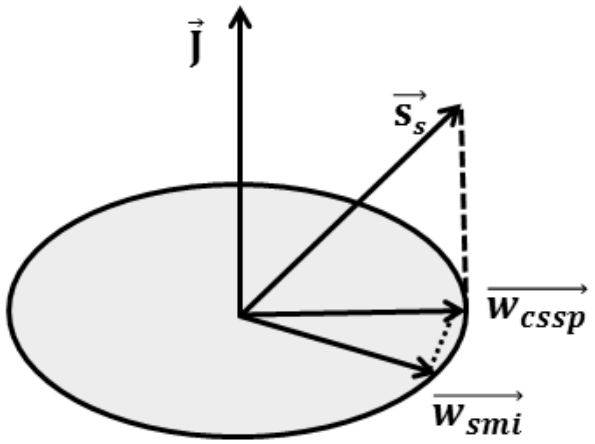


FIGURE 2.25 – Représentation du sous-espace perpendiculaire (disque gris) au sous-espace interférence $\vec{J}$, du *steering* vecteur $\vec{\mathbf{s}_\mathbf{s}}$ (poids au repos), et des poids adaptatifs, avec ($\vec{\mathbf{w}_{\mathbf{SMI}}}$) et sans CSSP ($\vec{\mathbf{w}_{\mathbf{CSSP}}}$)

2.3 Résultats

2.3.1 Simulation 1

Les performances sont testées sur les données simulées **SimALU** (voir Annexe A.3.3) qui simulent une antenne linéaire uniforme à 8 voies. La vitesse de la plateforme est de $V_a = 100m.s^{-1}$, la fréquence du radar est 10 GHz ($\lambda = 3.10^{-2}m$) et la PRF est fixée à $\frac{2V_a}{\lambda}$. A chaque rafale, le radar collecte 64 impulsions. Une cible de vitesse $v_t = 28ms^{-1}$ est ajoutée à la case distance de test. Nous comparons les différentes méthodes en comparant les détecteurs définis dans (1.5), (2.33), (2.45), ainsi que le SINR Loss qui est défini par

$$SINRLoss = \frac{\mathbf{w_q^H R_{th} w_q}}{\mathbf{w^H R w}} \frac{|\mathbf{w^H s}|^\mathbf{2}}{|\mathbf{w_q^H s}|^\mathbf{2}} \tag{2.53}$$

$\mathbf{w_q}$ est le vecteur de poids au repos, $\mathbf{R_{th}}$ est la matrice de covariance bruit seul (matrice identité dans notre cas) et où $\mathbf{w}$ est calculé par les différentes méthodes. Pour ces simulations, nous prenons une fenêtre temporelle $M = 6$ donc le nombre d'estimées $K_t = M_p - M + 1 = 59$ pour une taille de vecteur de $NM = 8 \times 6 = 48$. La ligne apparaissant sur les figures représentant la puissance en sortie est le seuil pour une $P_{fa} = 10^{-6}$.

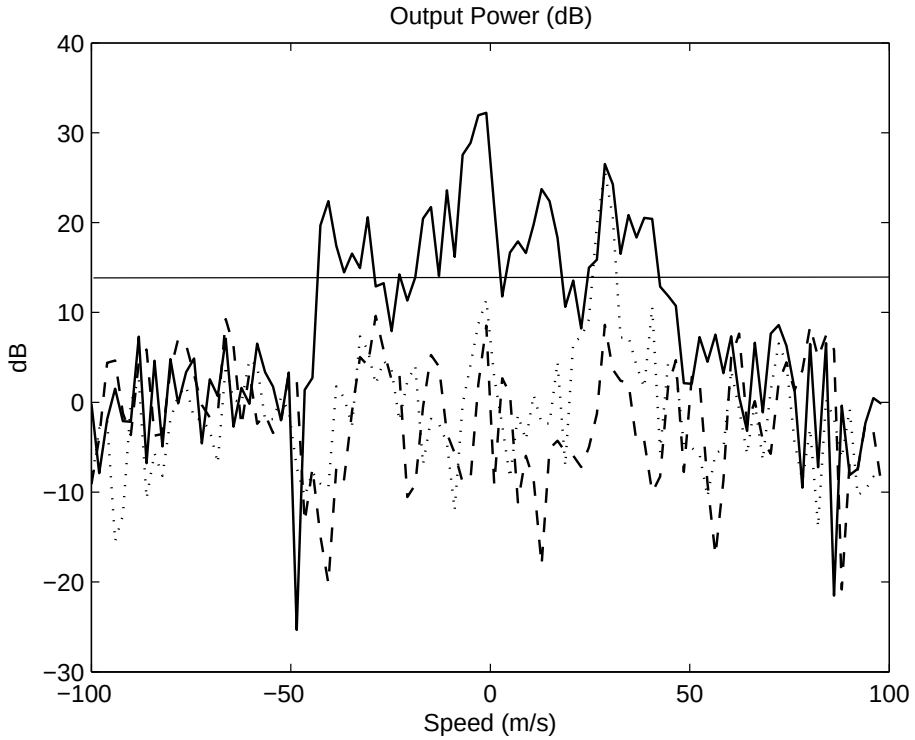


FIGURE 2.26 – Puissance SINR en sortie de la voie somme (solide), filtre optimal (pointillés) et méthode SMI classique (tirets)

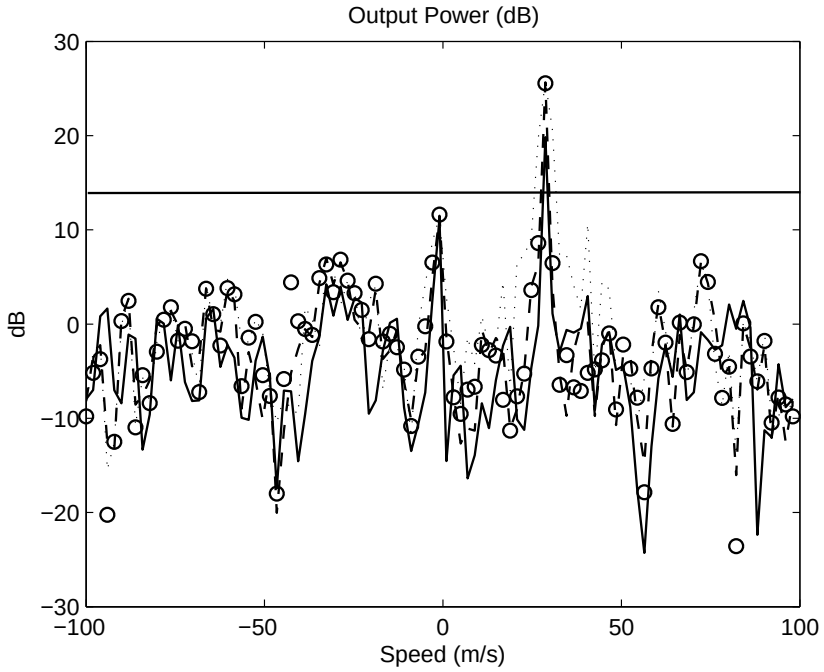


FIGURE 2.27 – Puissance SINR en sortie du MLED (solide), filtre optimal (pointillés), EC-APES (tirets) and FAPI-APES (cercles)

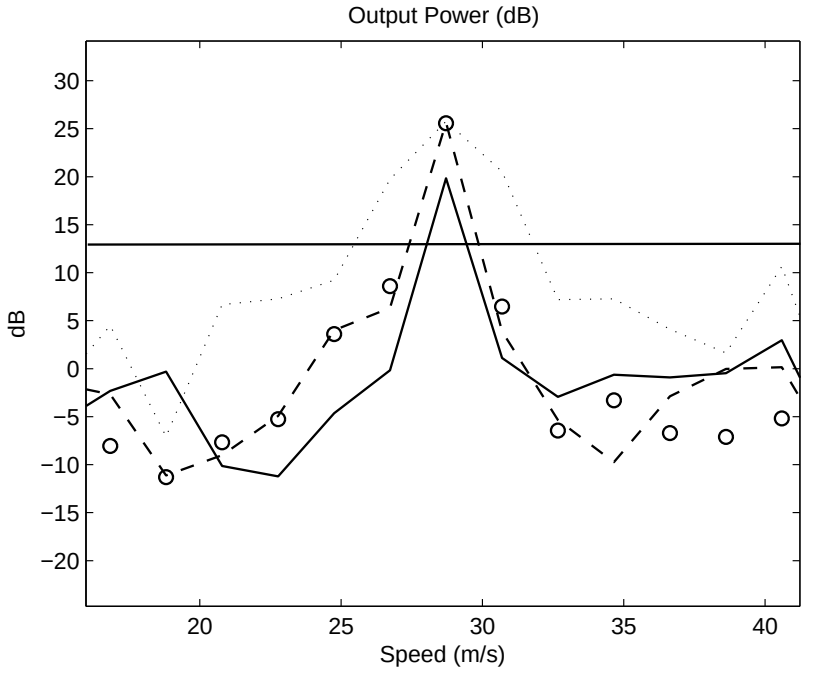


FIGURE 2.28 – Zoom sur la puissance SINR en sortie du filtre optimal (pointillés), MLED (solide), FAPI-APES (cercles) and EC-APES (tirets)

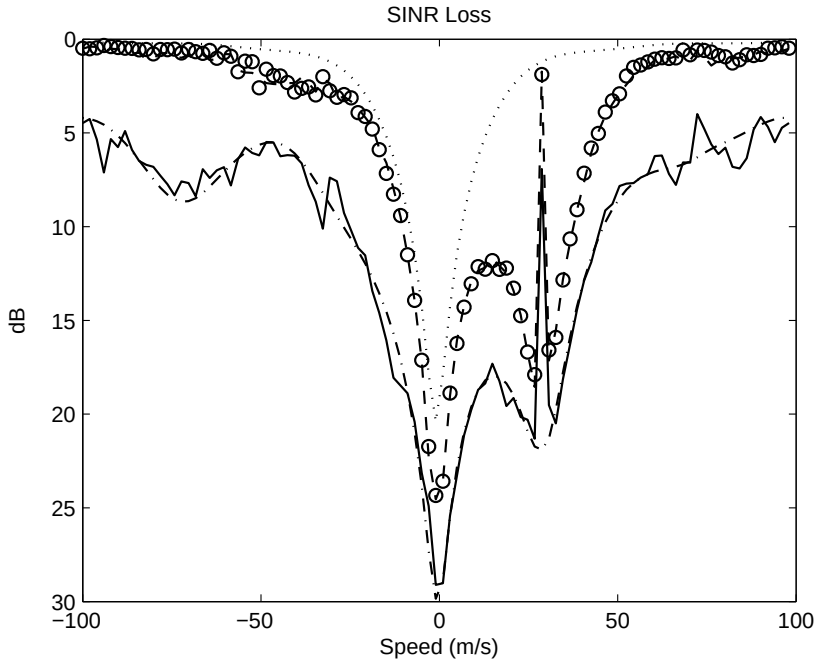


FIGURE 2.29 – SINR Loss du filtre optimal (pointillés), EC-APES (tirets), FAPI-APES (cercles), MLED (solide) et méthode SMI classique (tirets-pointillés)

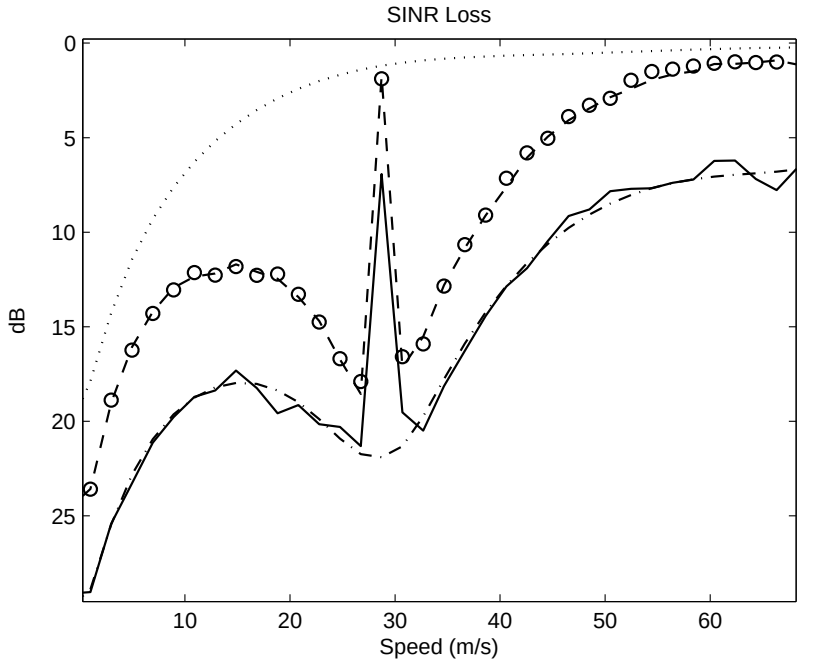


FIGURE 2.30 – Zoom sur le SINR Loss du filtre optimal (pointillés), EC-APES (tirets), FAPI-APES (cercles), MLED (solide) and méthode SMI classique (tirets-pointillés)

Les Figures 2.26 and 2.27 montrent le SINR en sortie des différents filtres en fonction de la vitesse (ou fréquence Doppler). Sur la Figure 2.26, nous pouvons voir le SINR en sortie du filtre STAP optimal et pour comparaison, le SINR en sortie de la voie somme (sans filtrage) avec le niveau du SINR en sortie du traitement STAP SMI classique (le fouillis est supprimé mais la cible l'est aussi). Les performances des SINR en sortie du MLED, EC-APES, FAPI-APES et du STAP optimal sont affichées sur la Figure 2.27. Nous pouvons voir que les algorithmes basés sur APES réussissent à atténuer le fouillis sans éliminer la cible. La Figure 2.28 est un zoom de la Figure 2.27 autour de la cible. Elle montre que les méthodes de sous-espace EC-APES et FAPI-APES surpassent le MLED avec un gain d'au moins 5 dB. Notons aussi que EC-APES et FAPI-APES donnent des résultats similaires alors que FAPI-APES demande une charge de calcul bien moindre que EC-APES (ceci sera vu dans la Partie III).

La Figure 2.29 montre le SINR Loss en fonction de la vitesse pour les filtres optimal, SMI, MLED, EC-APES et FAPI-APES. Nous pouvons voir que, comme le nombre d'échantillons K_t utilisés pour l'estimation de la matrice de covariance dans est inférieur à $2MN$, le SINR Loss pour SMI et MLED est d'environ 6dB de moins que le filtre optimal même dans les zones de vitesse hors fouillis et hors cible. Le MLED, quant à lui, permet de ne pas atténuer la cible alors que le SMI classique ne le permet pas. Cette figure montre que EC-APES et FAPI-APES non seulement conservent parfaitement la cible, mais conservent un SNIR Loss proche du filtre optimal. La Figure 2.30 est un zoom de la Figure 2.29 autour de la vitesse de la cible. Nous pouvons observer le gain des méthodes proposées EC-APES et FAPI-APES par rapport au MLED sur le SINR Loss à la vitesse de la cible.

2.3.2 Simulation 2

Nous présentons aussi les performances des méthodes de sous-espace sur les données synthétiques réalistes de **ONERA-AS** (voir Annexe A.3.1). Les paramètres sont les suivants : antenne à visée frontale 8 voies, fenêtre temporelle $M = 6$, $M_p = 64$ impulsions, $PRF = 2000Hz$, et vitesse du porteur $V_a = 180m.s^{-1}$. Nous choisissons de nous concentrer sur une zone où un convoi est présent. Les convois sont source de dégradation de performances pour les traitement STAP traditionnels parce que plusieurs cibles ayant la même vitesse à des cases distances différentes partagent une même direction d'arrivée. Leur présence dans les données d'entrainement mène à une atténuation de la cible. Ici, nous faisons aussi une simulation avec le détecteur MLED standard utilisant un *Diagonal Loading* d'un facteur $\delta = 1$, i.e $\mathbf{R_{DL}} = \mathbf{R} + \delta\mathbf{R_{th}}$, avec $\mathbf{R_{th}}$ étant la matrice de covariance estimée du bruit thermique (voir Annexe B.4). Sur la Figure 2.31, le convoi n'est pas visible en sortie de voie somme, seules les cibles exo-clutter sont visibles.

Sur la Figure 2.32, le convoi est clairement détectable aux distances 60-60.5 km en sortie de traitement STAP optimal, tout comme d'autres cibles à d'autres cases distance.

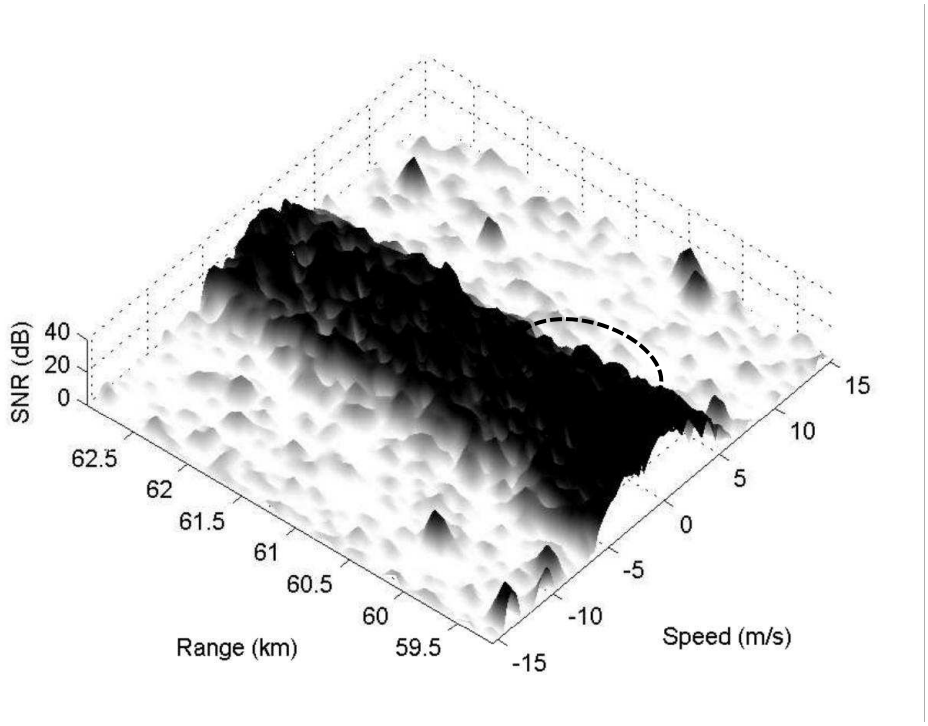


FIGURE 2.31 – SINR en sortie de la voie somme. Le convoi est entouré en tirets.

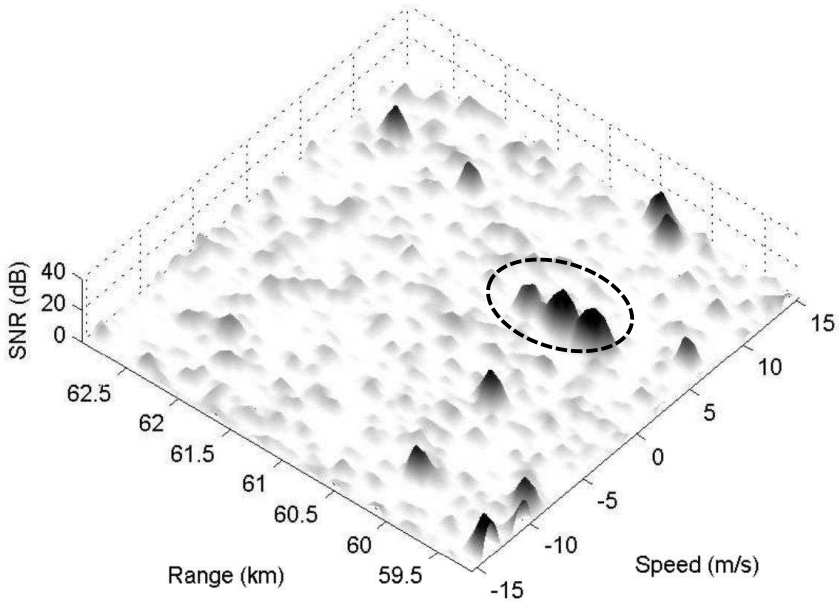


FIGURE 2.32 – SINR en sortie du traitement STAP optimal. Le convoi est entouré en tirets.

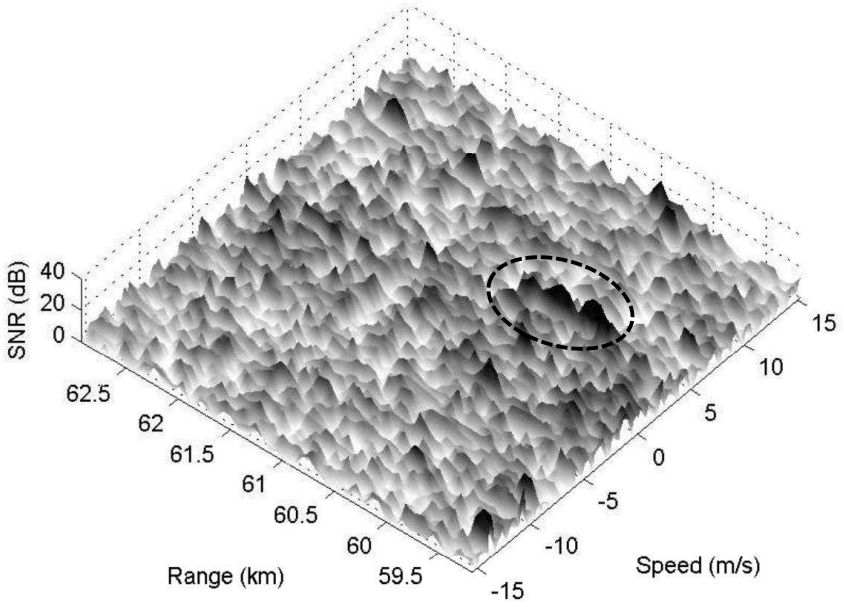


FIGURE 2.33 – SINR en sortie du MLED sans Diagonal Loading. Le convoi est entouré en tirets.

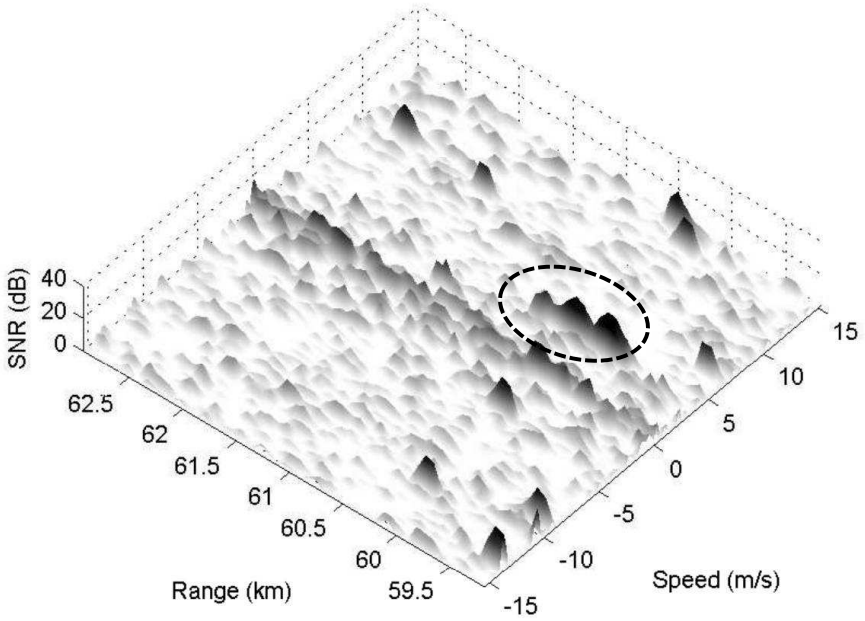


FIGURE 2.34 – SINR en sortie du MLED avec Diagonal Loading. Le convoi est entouré en tirets.

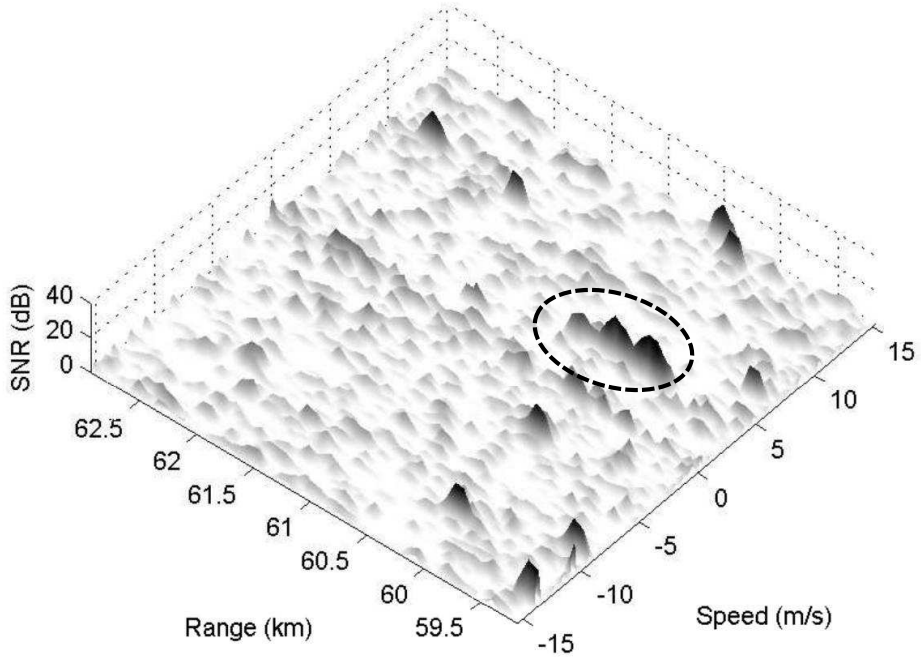


FIGURE 2.35 – SINR en sortie du traitement FAPI-APES. Le convoi est entouré en tirets.

Les résultats de la méthode MLED sans Diagonal Loading mettent en évidence une dégradation sévère du rapport signal à brut des cibles (voir Figure 2.33), et la détection n'est pas possible. Cependant, l'ajout d'un Diagonal Loading à la matrice de covariance, rend cette méthode plus robuste comme constaté sur la Figure 2.34. Pourtant, nous pouvons observer que le fouillis n'est pas complètement atténué aux vitesses faibles. Ce problème est résolu par les méthodes de sous-espace, comme le montre la 2.35.

2.4 Conclusion

Nous avons proposé deux versions plus robustes de l'algorithme MLED basées sur deux méthodes à rang réduit qui nécessitent moins de données d'entrainement pour l'estimation. Dans les situations hétérogènes, les méthodes traditionnelles font face à un manque de données d'entrainement en quantité suffisante ce qui entraine une dégradation des performance alors que nos méthodes maintiennent un niveau de performances satisfaisant même avec une quantité de données limitée. Les aspects négatifs des méthodes à rang réduit, c'est à dire le choix dans la séparation des sous-espaces interférence et bruit ainsi que la charge de calcul qui est plus élevée que pour les méthodes classiques d'inversion matricielle ont été surmontés. Ce dernier point est très important pour une application pratique. Notre nouvel algorithme FAPI-APES qui a les mêmes performances que EC-APES réduit significativement la charge de calcul par rapport à EC-APES, arrivant même à surpasser le MLED standard dans ce domaine.

3

Deterministic-aided STAP

Sommaire

Nous avons étudié dans les chapitres précédents différentes méthodes basées sur l'algorithme APES. Nous avons notamment remarqué que cet algorithme, si il permet d'utiliser uniquement les données primaires, a l'inconvénient de retirer une petite fraction du signal utile de la matrice de covariance. Dans ce chapitre, nous mettons en évidence cet effet puis nous étudions une approche déterministe du traitement spatio-temporel. Nous proposons ensuite deux méthodes de traitement adaptatif, qui en s'aidant de cette approche déterministe, permettent de résoudre en partie ce problème lié à l'utilisation de la méthode APES.

3.1 Limites des méthodes basées sur APES

Dans le Chapitre 2, nous avons montré que la matrice de covariance $\mathbf{Q}$ peut s'écrire sous la forme :

$$\mathbf{Q} = \frac{\mathbf{N}\mathbf{N}^{\mathbf{H}}}{K_t} - \frac{\mathbf{N}\mathbf{s}_{\mathbf{t}}^{*}\mathbf{s}_{\mathbf{t}}^{\mathbf{T}}\mathbf{N}^{\mathbf{H}}}{K_t^2} \tag{3.1}$$

où la matrice $\frac{\mathbf{N}\mathbf{N}^{\mathbf{H}}}{K_t}$ est la matrice de covariance estimée des interférence et du bruit alors que $\frac{\mathbf{N}\mathbf{s}_{\mathbf{t}}^{*}\mathbf{s}_{\mathbf{t}}^{\mathbf{T}}\mathbf{N}^{\mathbf{H}}}{K_t^2}$ est le produit scalaire des vecteurs interférence plus bruit $\mathbf{N}$ avec leurs projections sur $\mathbf{s}_{\mathbf{t}}^{*}\mathbf{s}_{\mathbf{t}}^{\mathbf{T}}\mathbf{N}$. Nous avions noté que la matrice résiduelle contenant les interférences et le bruit est légèrement différente de la vraie matrice de covariance estimée $\frac{\mathbf{N}\mathbf{N}^{\mathbf{H}}}{K_t}$. Le terme $\frac{\mathbf{N}\mathbf{s}_{\mathbf{t}}^{*}\mathbf{s}_{\mathbf{t}}^{\mathbf{T}}\mathbf{N}^{\mathbf{H}}}{K_t^2}$ représente en fait la bande de fouillis et de bruit qui est retirée de la matrice de covariance lors du traitement APES. La Figure 3.1 permet de représenter ce phénomène de projection dans le plan angle-Doppler. Ce terme $\frac{\mathbf{N}\mathbf{s}_{\mathbf{t}}^{*}\mathbf{s}_{\mathbf{t}}^{\mathbf{T}}\mathbf{N}^{\mathbf{H}}}{K_t^2}$ s'écrit dans le cas Stop-Band APES $\frac{\mathbf{N}\mathbf{S}_{\mathbf{t}}^{*}\mathbf{S}_{\mathbf{t}}^{\mathbf{T}}\mathbf{N}^{\mathbf{H}}}{K_t^2}$ où $\mathbf{S}_{\mathbf{t}}^{\mathbf{T}}$ est définie en Section 1.3.

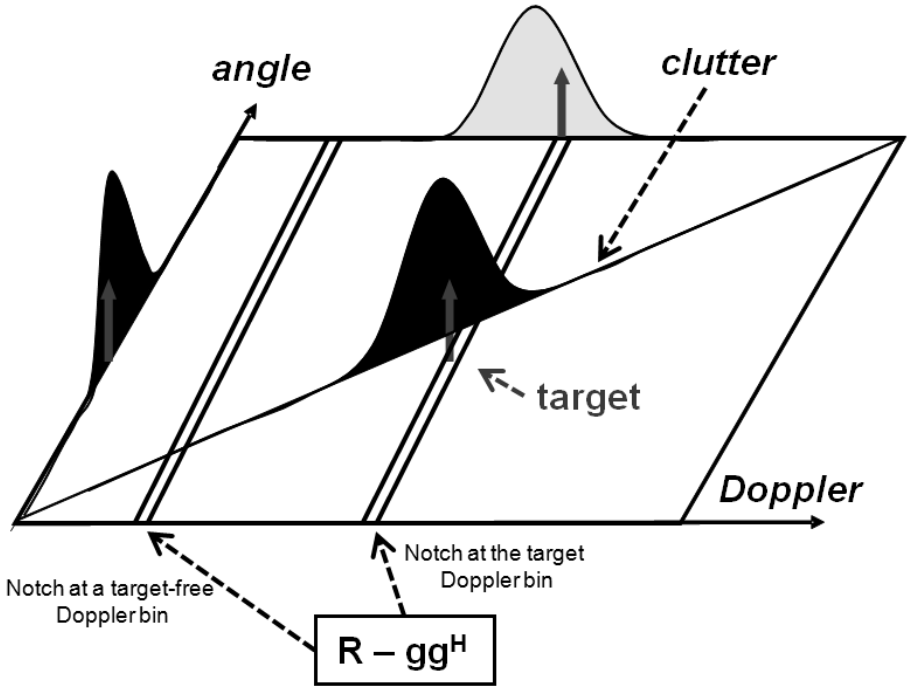


FIGURE 3.1 – Image angle-Doppler montrant l'effet du projecteur MLED pour deux cases Doppler différentes

La largeur spectrale du projecteur Stop-Band étant supérieure à celle de APES classique, ce terme sera plus important, diminuant légèrement les performances du filtre en réjection de fouillis.

Le nombre de cases Doppler étant généralement important, cette bande fréquentielle est en conséquence très fine dans le cas de Stop-Band APES et encore plus dans le cas APES standard, les pertes en réjection de fouillis peuvent alors être considérées comme négligeables. Cependant, dans les situations où le nombre de cases fréquentielle est faible, nous observerons une dégradation de la réjection du fouillis pour le détecteur MLED, et cette dégradation sera encore plus forte pour l'algorithme Stop-Band APES.

3.2 Approche déterministe

Nous décrivons ici un traitement spatio-temporel purement déterministe servant de base aux méthodes que nous présenterons dans les sections suivantes. Pour une antenne linéaire uniforme à visée latérale, la fréquence Doppler du fouillis est fonction de l'angle d'arrivée du signal :

$$f = \frac{2V}{\lambda}\sin\theta \;\Rightarrow\; \theta = \sin^{-1}\left(\frac{\lambda f}{2V}\right) \tag{3.2}$$

avec f la fréquence Doppler du fouillis, θ l'angle d'arrivée, V la vitesse du porteur et λ la longueur d'onde du signal émis. Connaissant cette relation, nous pouvons construire un filtre qui supprime le signal contenu dans ce domaine mono-dimensionnel régit par l'Eq. (3.2). La forme générale du filtre, qui sera appelé traitement déterministe, a toujours la même forme

$$\mathbf{w^H} = \frac{\mathbf{s_s^H K^{-1}}}{\mathbf{s_s^H K^{-1} s_s}} \tag{3.3}$$

mais avec

$$\mathbf{K} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{s_c}(\theta_i(f_i), f_i)\mathbf{s_c^H}(\theta_i(f_i), f_i) + \mathbf{\Gamma_N}$$

où $\mathbf{\Gamma_N}$ est la vraie matrice de covariance de bruit, k est le nombre de *patches* de fouillis, $\mathbf{s_c}(\theta_i(f_i), f_i)$ est le *steering* vecteur spatio-temporel d'angle θ_i et de fréquence f_i obtenu avec (3.2). Si l'on adopte la même formulation que dans le cas du filtre MLED et du Stop-Band APES, la matrice peut être écrite pour chaque case Doppler :

$$\mathbf{K'} = \frac{1}{k'}\sum_{i=1}^{k'} \mathbf{S_c}(\theta_i(f_i), f_i)\mathbf{S_c^H}(\theta_i(f_i), f_i) + \mathbf{\Gamma_N} \tag{3.4}$$

le vecteur $\mathbf{S_c}(\theta_i(f_i), f_i)$ est le *steering* vecteur théorique du fouillis obtenu avec la formule (3.2) et (1.11). Pour traiter une case Doppler, le steering vecteur $\mathbf{S_c}(\theta_0(f_0), f_0)$ de la case Doppler sous test et les deux steering vecteurs $\mathbf{S_c}(\theta_{\pm1}(f_{\pm1}), f_{\pm1})$ des cases Doppler voisines sont suffisants pour correctement supprimer le fouillis. Cependant, les performances de ce traitement non-adaptatif sont limitées en pratique à cause de l'hétérogénéité du fouillis (ex : milieux urbains, reliefs, etc...) et à cause des erreurs de phases induites par les récepteurs des antennes. Ces erreurs font que le vrai *steering* vecteur est légèrement différent du *steering* vecteur théorique utilisé avec la formule (3.2) pour construire la matrice de covariance.

Résultats

Pour illustrer cet effet, nous comparons le traitement non-adaptatif (3.3) à un traitement adaptatif FIR classique sur deux jeux de données. Les premières données **SimALU** sont des données que nous simulons en utilisant le vrai *steering* vecteur théorique alors que le deuxième jeu de données sont les données **ONERA-ALU** (voir Annexe A.3.1) provenant d'un simulateur réaliste qui simule des erreurs de phases sur les voies de réceptions et ajoute aléatoirement des échos impulsionnels dans le fouillis. Dans les deux cas, le fouillis est Gaussien, homogène et a une puissance de 40dB en sortie sur la voie somme. L'antenne est linéaire uniforme à visée latérale. Aucune cible n'est présente dans ces données.

Le traitement non-adaptatif est seulement appliqué sur le domaine des vitesses positives, c'est à dire que le domaine des vitesses négatives est la voie somme. Le traitement adaptatif est quant à lui appliqué sur l'ensemble des vitesses.

Comme nous pouvons le voir sur les Figures 3.2 et 3.3, le traitement non-adaptatif fonctionne bien sur les données simulées ne contenant pas d'erreurs. Le rapport fouillis à bruit (CNR) est proche de 0dB, comme pour le traitement adaptatif (c.f Figures 3.2 et 3.3), ce qui implique une atténuation d'environ 40dB. Sur les données réalistes, le traitement non-adaptatif n'arrive pas à correctement supprimer le fouillis (c.f Figures 3.5 et 3.6). En effet, le CNR résiduel est d'environ 15dB c'est à dire une atténuation de 25dB, entrainant un nombre de fausses alarmes élevé. Les Figures 3.5 et 3.6 montrent cet effet. Nous déduisons de ces résultats que nous ne pouvons pas utiliser un traitement déterministe dans des situations réalistes. En revanche il est possible de tirer profit de la relation déterministe angle-Doppler du fouillis dans un traitement adaptatif pour améliorer ses performances, ce que nous voyons sur la Figure 3.7.

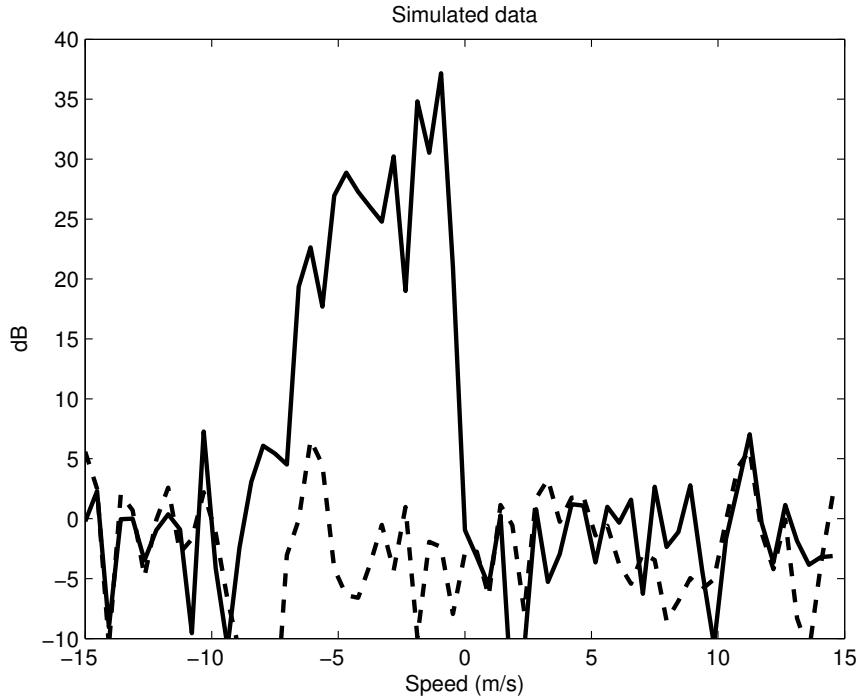


FIGURE 3.2 – Comparaison entre la voie somme (gras, vitesses négatives), déterministe (gras, vitesses positives) et adaptatif (tirets) sur les données sans erreurs

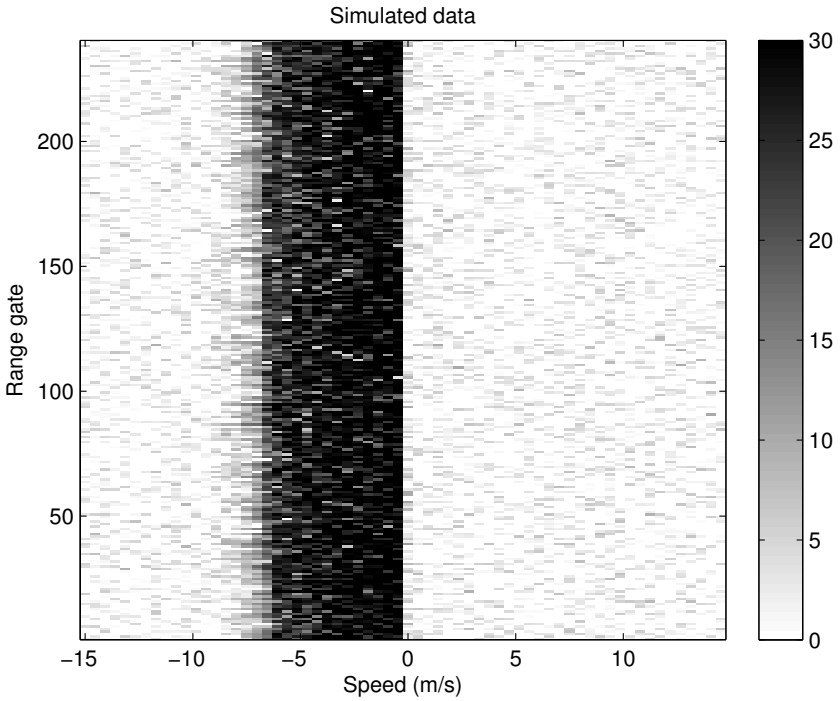


FIGURE 3.3 – Image vitesse-distance du traitement non-adaptatif (vitesse positives) et voie somme (vitesses négatives) sur les données sans erreurs

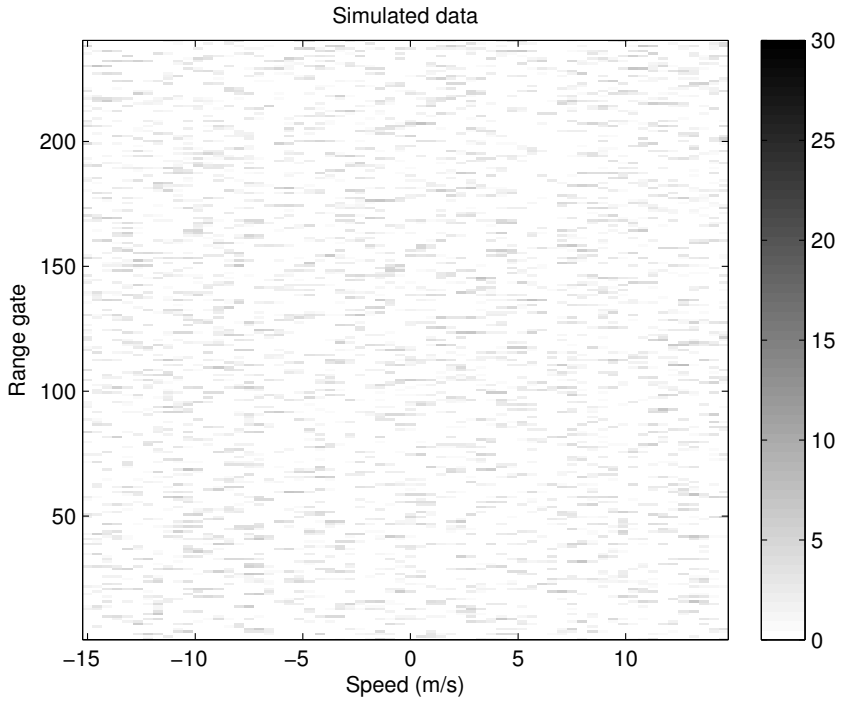


FIGURE 3.4 – Image vitesse-distance du traitement adaptatif sur les données sans erreurs

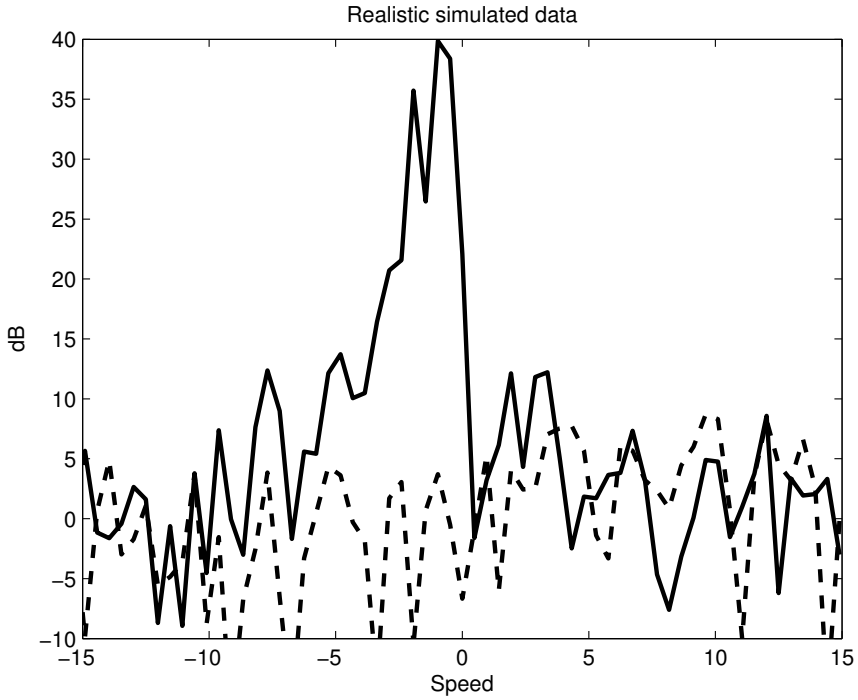


FIGURE 3.5 – Comparaison entre voie somme (gras, vitesses négatives), traitement déterministe (gras, vitesses positives) et adaptatif (tirets) sur données réalistes

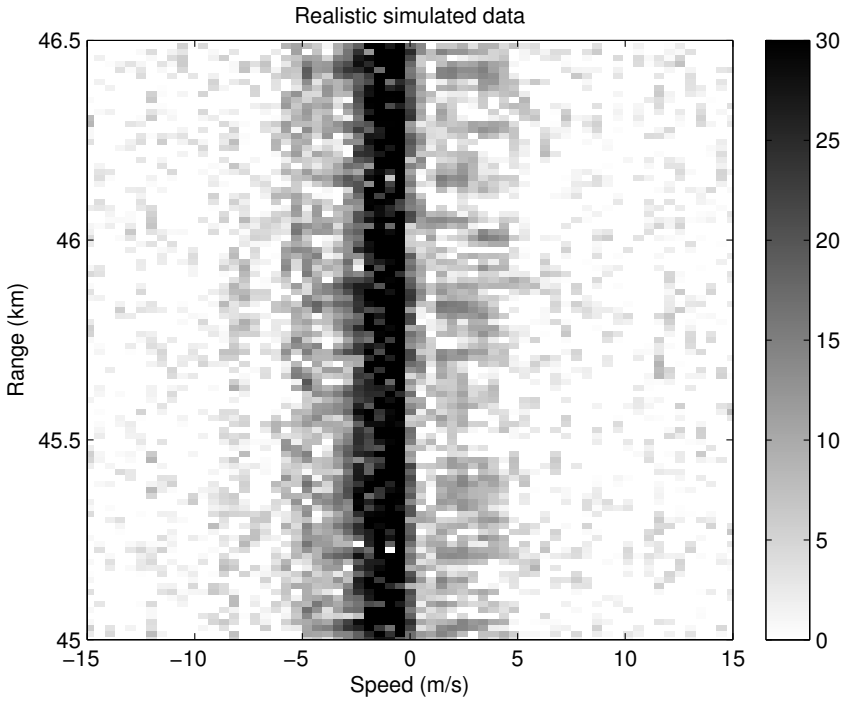


FIGURE 3.6 – Image vitesse-distance du traitement déterministe (vitesses positives) et voie somme sur données réalistes

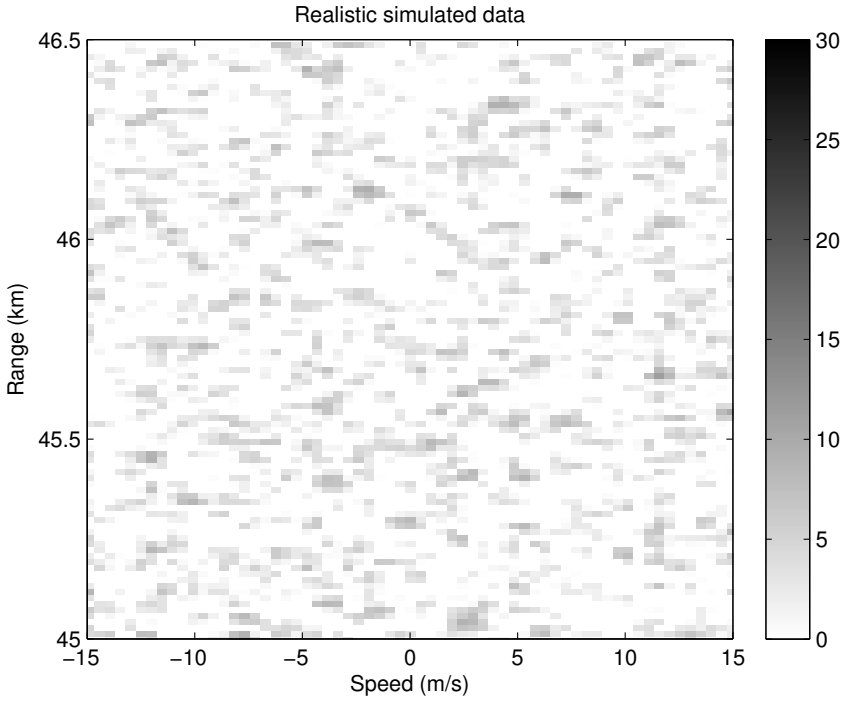


FIGURE 3.7 – Image vitesse-distance du traitement adaptatif sur données réalistes

3.3 Deterministic-aided STAP

3.3.1 Deterministic-aided GMTI STAP

L'utilisation de méthodes utilisant des données primaires seules peut être très utile en milieu fortement hétérogène (voir Chapitre 1). Pour résoudre les limitations liées à l'utilisation de méthodes basées sur l'algorithme APES (voir Section 3.1), nous proposons une nouvelle méthode s'aidant de certaines propriétés du traitement déterministe étudié dans la Section précédente.

Nous avons vu dans l'équation (3.1) que le terme $\mathbf{gg^H}$ défini en (1.3) retire le signal d'intérêt (s'il est présent) mais aussi une petite partie du signal interférence. L'idée ici est d'essayer de ré-injecter ce signal fouillis dans la matrice de covariance $\mathbf{Q}$ afin d'obtenir une nouvelle matrice $\mathbf{T}$:

$$\mathbf{T} = \mathbf{R} - \mathbf{gg^H} + \mathbf{g_c g_c^H} \tag{3.5}$$

où $\mathbf{g_c}$ est la projection de $\mathbf{g}$ sur le *steering* vecteur théorique du fouillis $\mathbf{s_c}(\theta(f), f)$ calculé avec la formule déterministe de l'équation (3.2) :

$$\mathbf{g_c} = \mathbf{P_c g} \tag{3.6}$$

avec

$$\mathbf{P}_c = \frac{\mathbf{s_c}(\theta(f), f)\mathbf{s_c}^H(\theta(f), f)}{\mathbf{s_c}^H(\theta(f), f)\mathbf{s_c}(\theta(f), f)}$$

Nous pouvons démontrer que la nouvelle matrice de covariance $\mathbf{T}$ de (3.5) peut être maintenant écrite :

$$\mathbf{T} = \frac{\mathbf{NN^H}}{K_t} - \frac{\mathbf{N s_t^* s_t^T N^H}}{K_t^2} + \frac{\mathbf{P_c X s_t^* s_t^T X^H P_c^H}}{K_t^2} \tag{3.7}$$

Si le fouillis suit la relation théorique angle-Doppler de (3.2), alors la projection du signal sur le *steering* vecteur sera proche du signal fouillis qui a été retranché de la matrice de covariance (i.e $(\mathbf{P_c X} \approx \mathbf{N})$, et la matrice de covariance $\mathbf{T}$ sera proche de

$$\mathbf{T} \approx \frac{\mathbf{NN^H}}{K_t} \tag{3.8}$$

Notons qu'il n'y a pas besoin de fixer une puissance arbitraire de fouillis parce que la puissance est contenue dans $\mathbf{g_c}$ (cf. (3.6)).
Dans le cas du Stop-Band APES, où le projecteur APES est plus large, nous utilisons un projecteur étendu : $\mathbf{P_c}$:

$$\mathbf{P}_c = \frac{\mathbf{S_c S_c^H}}{\mathbf{S_c^H S_c}}$$

avec

$$\mathbf{S_c} = [\mathbf{s_c}(\theta, f - \frac{\Delta f}{2})\mathbf{s_c}(\theta, f)\mathbf{s_c}(\theta, f + \frac{\Delta f}{2})]$$

où θ et f sont encore liés par la relation de l'équation (3.2).

3.3.2 Deterministic-aided STAP pour mode air-to-air

En situations air-air, le problème est différent. L'occupation spectrale du fouillis arrivant par le lobe principal est plus faible qu'en mode GMTI, alors que le fouillis arrivant des lobes secondaires est plus fort et doit être supprimé. De plus, la densité de cibles est beaucoup plus faible qu'en mode air-sol. Dans ce cas nous n'avons pas de relation théorique angle-Doppler pour le fouillis, il n'est donc pas possible d'appliquer la méthode précédente.

Nous proposons une autre approche pour ré-injecter le fouillis partiellement retiré par l'algorithme APES. En mode air-air, le fouillis arrivant par le lobe principal est homogène en distance. Nous allons exploiter cette propriété pour estimer une matrice $\mathbf{g_c g_c^H}$. Le domaine d'estimation de cette matrice est le domaine distance. Pour chaque case Doppler, la matrice de covariance $\mathbf{T}$ est définie par :

$$\mathbf{T} = \mathbf{R} - \mathbf{g}\mathbf{g}^{\mathbf{H}} + \mathbf{C} \tag{3.9}$$

Cependant, la matrice $\mathbf{C}$ qui est égale à $\mathbf{g_c g_c^H}$ dans (3.5) est maintenant estimée de la façon suivante pour chaque case Doppler :

$$\mathbf{C} = \sum_{i=1}^{L} \frac{1}{L} \mathbf{g}_i \mathbf{g}_i^{\mathbf{H}} \tag{3.10}$$

$\mathbf{g_i}$ étant le vecteur $\mathbf{g_i} = \frac{\mathbf{X_i s_t^*}}{K_t}$ de la case distance i, et L est le nombre total de cases distance. Le fouillis étant homogène, nous pouvons alors faire l'approximation suivante :

$$\mathbf{C} = \frac{1}{L} \sum_{i=1}^{L} \mathbf{g}_i \mathbf{g}_i^{\mathbf{H}} \approx \mathbf{g_c g_c^H}$$

Cette supposition implique que seule la partie homogène du fouillis sera ré-injectée dans la matrice de covariance. La densité de cible doit rester faible, sinon du signal d'intérêt sera ré-injecté dans la matrice de covariance de manière non négligeable et le rapport signal à bruit des cibles sera atténué. Dans le cas de Stop-Band APES, la matrice $\mathbf{Q}$ de l'équation (1.16) est remplacée par la matrice $\mathbf{T}$:

$$\mathbf{T} = \frac{\mathbf{X}\mathbf{X}^H}{\mathbf{s_t^T s_t^*}} - \frac{1}{K_t} \mathbf{X} \mathbf{P}_{//} \mathbf{X}^H + \frac{1}{L} \sum_{i=1}^{L} \mathbf{X}_i \mathbf{P}_{//} \mathbf{X}_i^H \tag{3.11}$$

avec $\mathbf{X}$ les données de la case sous test, $\mathbf{P}_{//}$ le projecteur Stop-Band défini dans (1.10), $\mathbf{X_i}$ les données de la case distance i et L est le nombre total de cases distance. Cette méthode de ré-injection de fouillis suppose qu'une partie du fouillis que l'on veut re-injecter dans la matrice de covariance soit stationnaire et que la densité de cibles soit faible. Elle est donc adaptée aux modes Air-air plutôt qu'aux modes GMTI.

3.4 Résultats

3.4.1 Simulations GMTI

Nous testons le GMTI Deterministic-Aided STAP décrit en Section 3.3.1 sur les données réelles **SAREX** (voir Annexe A.3.4). Nous rappelons les paramètres de ces données : antenne ALU 4 voies, vitesse de l'avion $V_a = 85m.s^{-1}$, fréquence de répétition PRF $= 1.5625\ kHz$, 300 cases distance et 64 impulsions. Le nombre d'échantillons temporels choisi pour former le vecteur spatio-temporel a été fixé à $Ktaps = 7$. Trois cibles sont présentes dans cette scène.

	cible 1	cible 2	cible 3
Vitesse (m/s)	3.0	5.30	5.85
Distance (numéro)	214	138	149

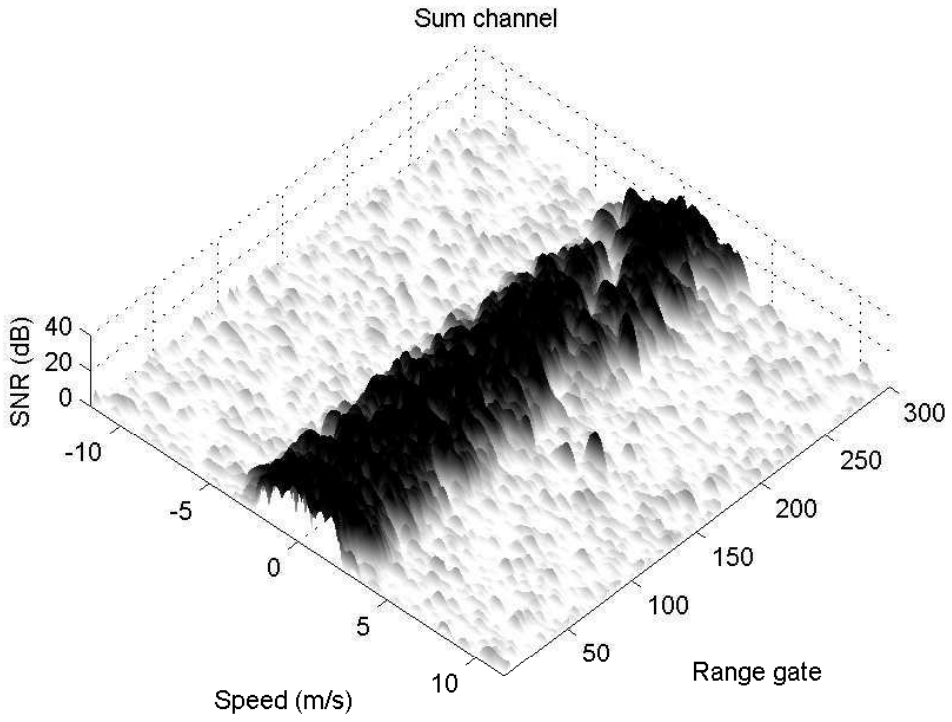


FIGURE 3.8 – Image Doppler-distance de la voie somme

L'image Doppler distance de la voie somme (Fig. 3.8) montre l'hétérogénéité du fouillis, les cibles 2 et 3 sont détectables sans traitement alors que la cible 1 est noyée dans le fouillis. Les figures suivantes présentent les résultats du traitement STAP classique avec estimation sur 20 cases distance et 2 cases de garde, du traitement MLED, du Stop-Band APES et du Deterministic-Aided Stop-Band STAP (estimation sur 3 cases distances sans case de garde pour tous ces derniers traitements). Aucun sur-échantillonnage n'est fait pour ces traitements (bien qu'un 2x zero-padding soit requis pour le Stop-Band APES) sauf pour le traitement MLED où nous utilisons un sur-échantillonnage 4x.

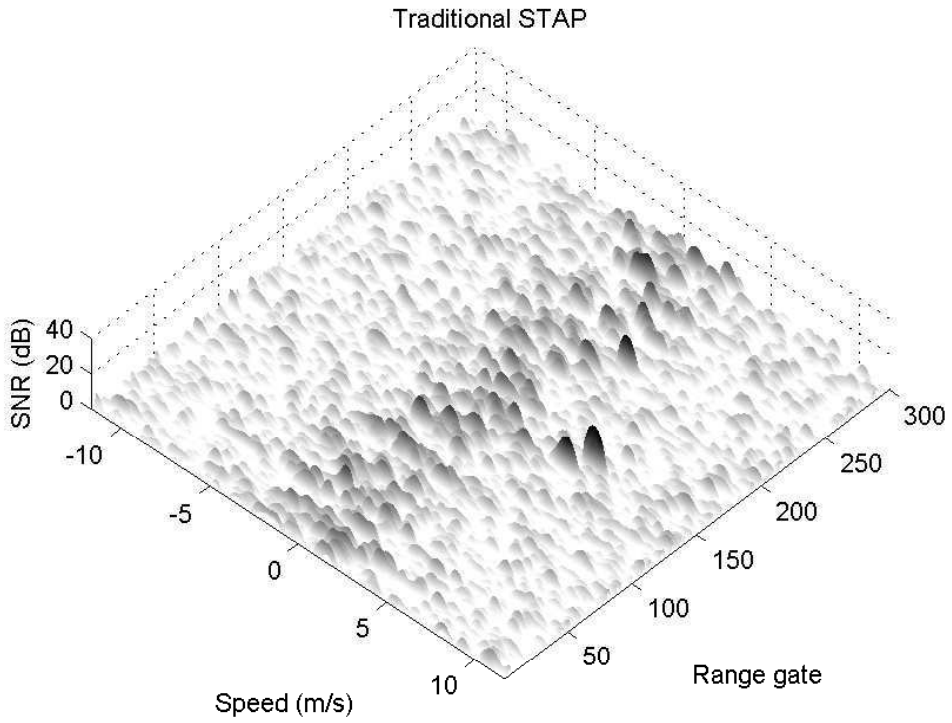


FIGURE 3.9 – Image Doppler-distance : performances du traitement STAP classique

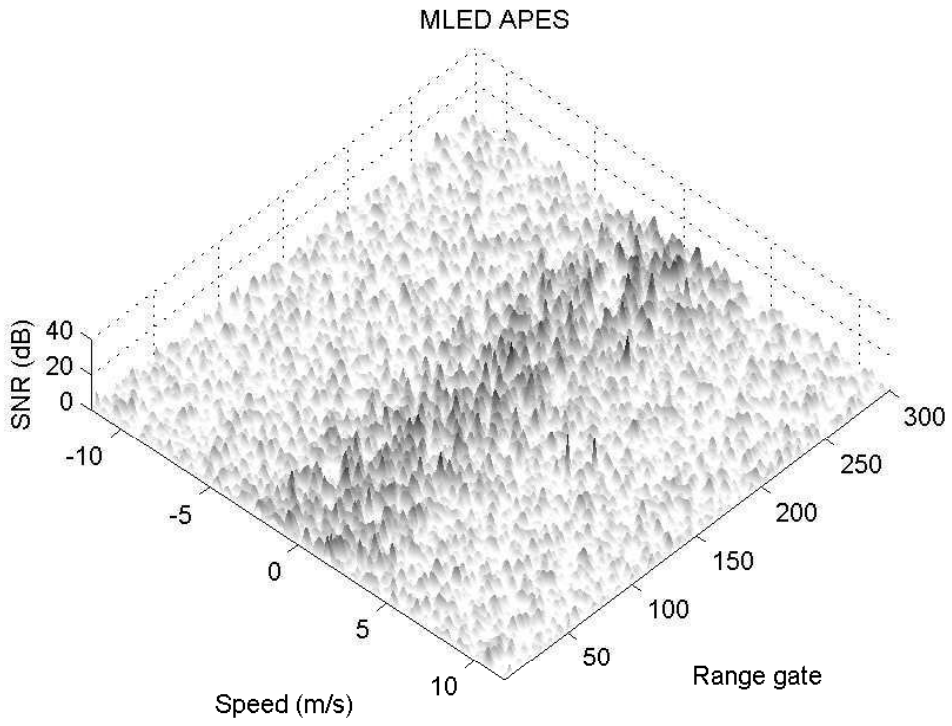


FIGURE 3.10 – Image Doppler-distance : performances du traitement MLED

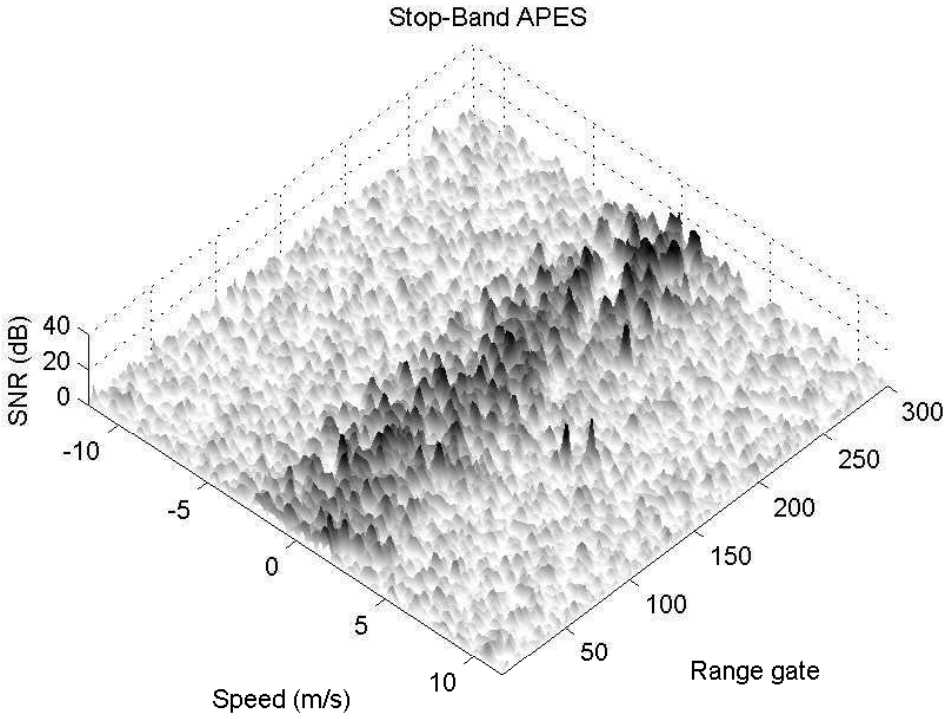


FIGURE 3.11 – Image Doppler-distance : performances du traitement Stop-Band APES

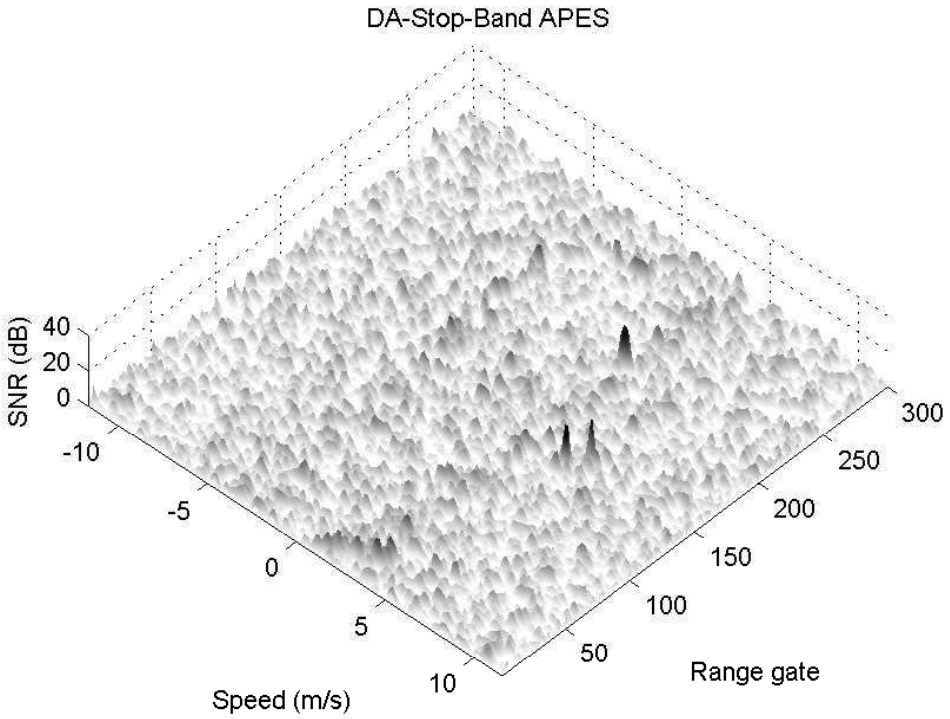


FIGURE 3.12 – Performances du traitement Deterministic-Aided Stop-Band APES

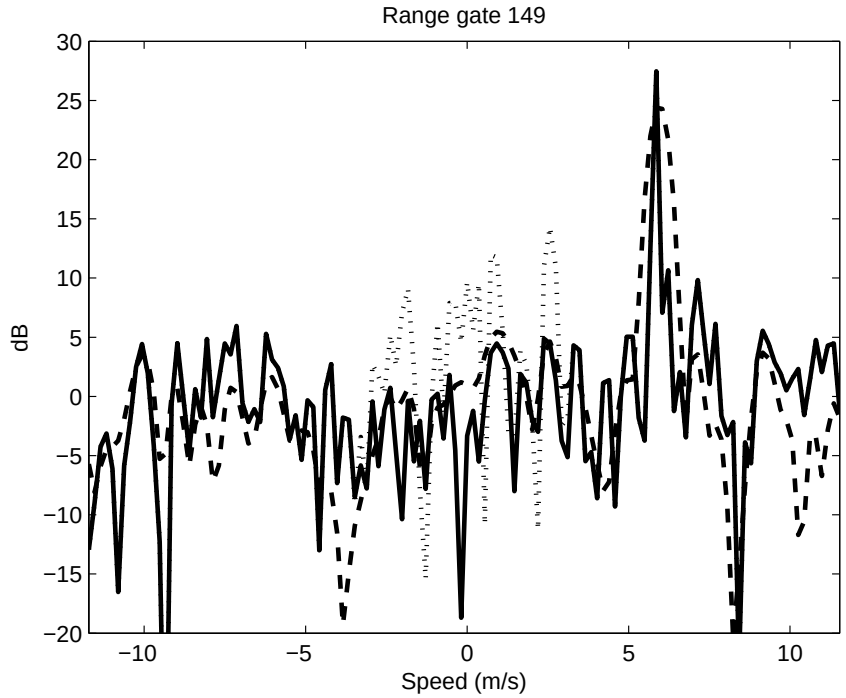


FIGURE 3.13 – Comparaison : STAP classique (tirets) Stop-Band APES (pointillés) et Deterministic-Aided Stop-Band (solide) pour la case distance 149 (cible numéro 3)

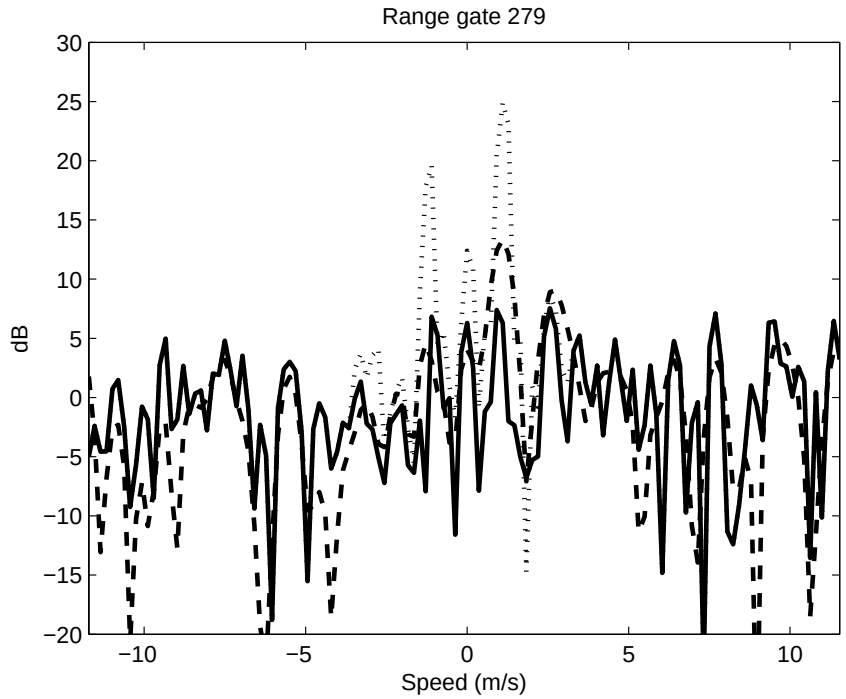


FIGURE 3.14 – Comparaison : STAP classique (tirets) Stop-Band APES (pointillés) et Deterministic-Aided Stop-Band (solide) pour la case distance 279 (pas de cible)

Le traitement STAP classique ne parvient pas à correctement supprimer le fouillis hétérogène (Fig. 3.9). Le détecteur MLED dont le projecteur est très fin ne parvient pas non plus à complètement supprimer le signal. A cause de sa propriété d'hyper-résolution, l'étalement Doppler des cibles est très étroit, ce qui les rend difficiles à distinguer sur l'image Doppler-distance de la Figure 3.10.

Comme prévu, le traitement Stop-Band APES laisse passer encore plus de fouillis comme nous pouvons l'observer sur la Figure 3.11, alors que le Deterministic-Aided Stop-Band (DA-Stop-Band) supprime efficacement le fouillis (Fig. 3.12). Aucune atténuation n'est constatée sur la cible 1 qui est dans la bande Doppler du fouillis. La Figure 3.13 montre l'amélioration de l'atténuation du fouillis du DA-Stop-Band sur le Stop-Band pour la case distance 149, où la cible 3 est présente. La Figure 3.14 montre la supériorité du DA-Stop-Band sur le traitement STAP classique en réjection de fouillis à la case distante 279, une zone où le fouillis est particulièrement puissant.

3.4.2 Simulations Air-air

Le traitement *Deterministic-Aided* pour le mode air-air (voir Section 3.3.2) est testé sur les données réalistes simulées **ONERA-AA** (voir Annexe A.3.1). Nous rappelons les caractéristiques de ces données : l'antenne est de type AMSAR, la vitesse de l'avion est $V_a = 300m.s^{-1}$, la fréquence de répétition est PRF $= 20\ kHz$, le nombre de cases distance est 100 correspondant à un intervalle de distance de 52.5 km à 59.5 km de portée. Le nombre d'échantillons temporels utilisés pour former le vecteur STAP est $Ktaps = 8$ et le nombre d'impulsions total est 128. Cinq cibles sont présentes dans la scène (voir tableau ci-dessous).

Cibles	Vitesse (m/s)	Distance (km)
1	50.03	58.425
2	100.22	55.425
3	115.026	57.00
4	185.0265	57.30
5	216.0296	59.475

La voie somme (Fig.3.15) montre clairement que le fouillis du lobe principal (vitesses de 230 à 280 m/s) et le fouillis des lobes secondaires occupent une partie importante de l'image Doppler-distance. Seulement deux cibles sont détectables sans traitement STAP. Sur la Figure 3.16, nous pouvons voir que le traitement STAP classique supprime le fouillis homogène du lobe principal avec succès mais ne supprime pas le fouillis arrivant par les lobes secondaires qui est très hétérogène.

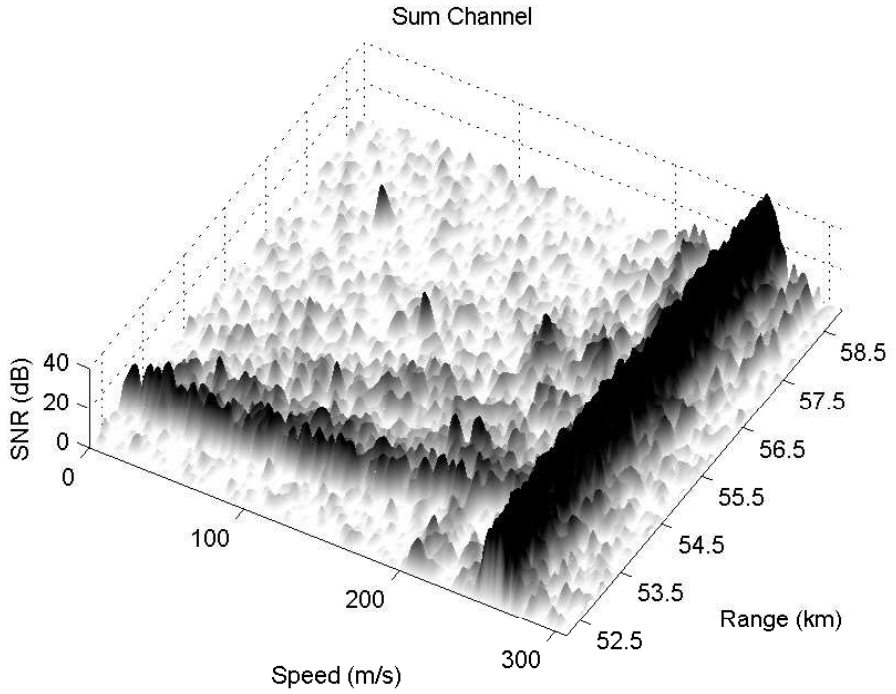


FIGURE 3.15 – Image Doppler-distance mode air-air : voie somme

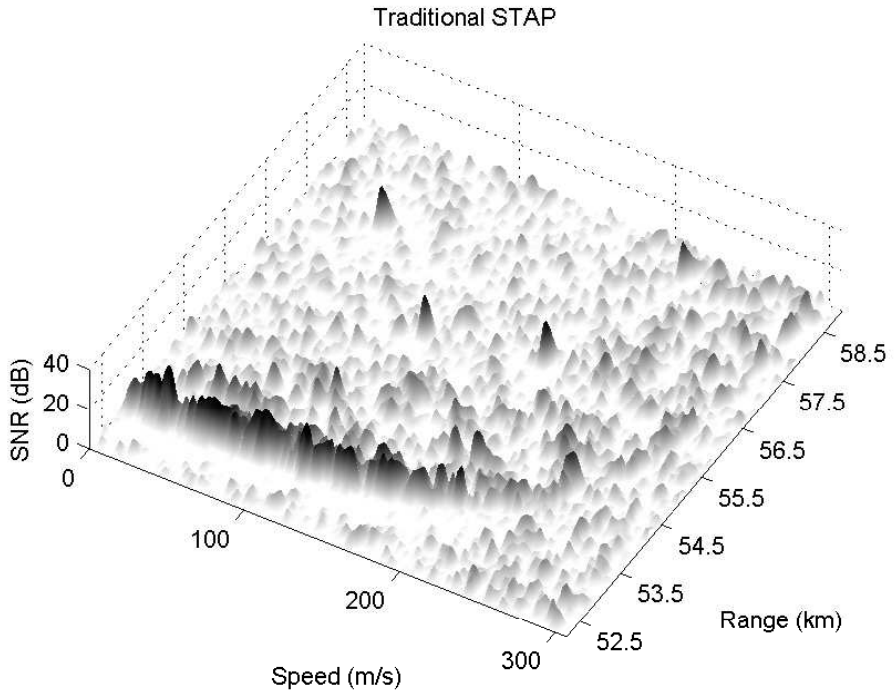


FIGURE 3.16 – Mode air-air : traitement STAP FIR (référence)

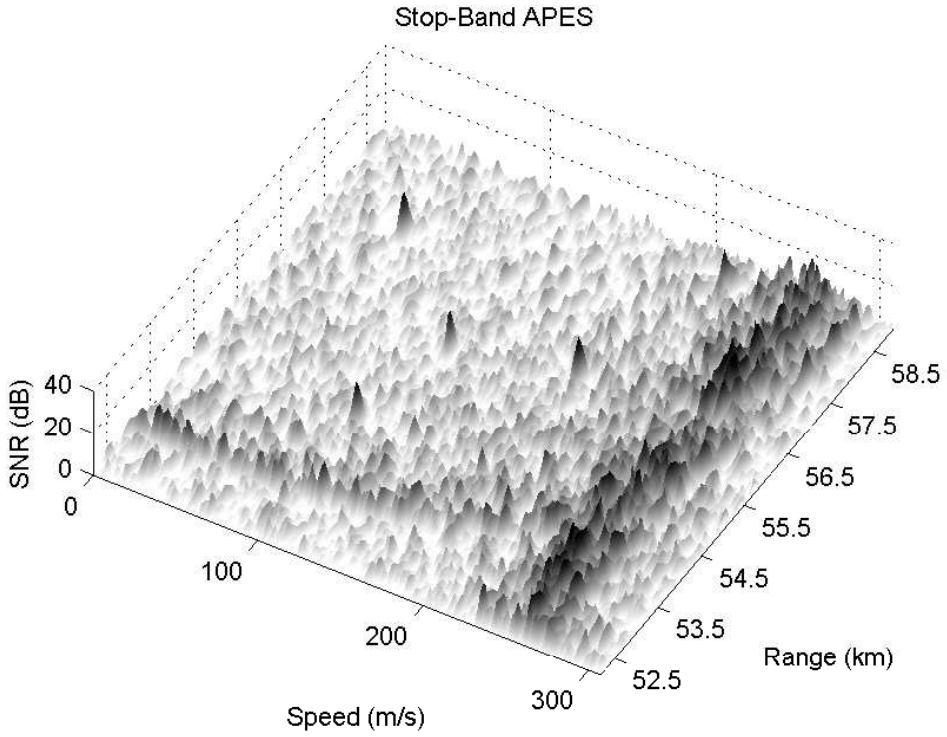


FIGURE 3.17 – Mode air-air : traitement Stop-Band APES

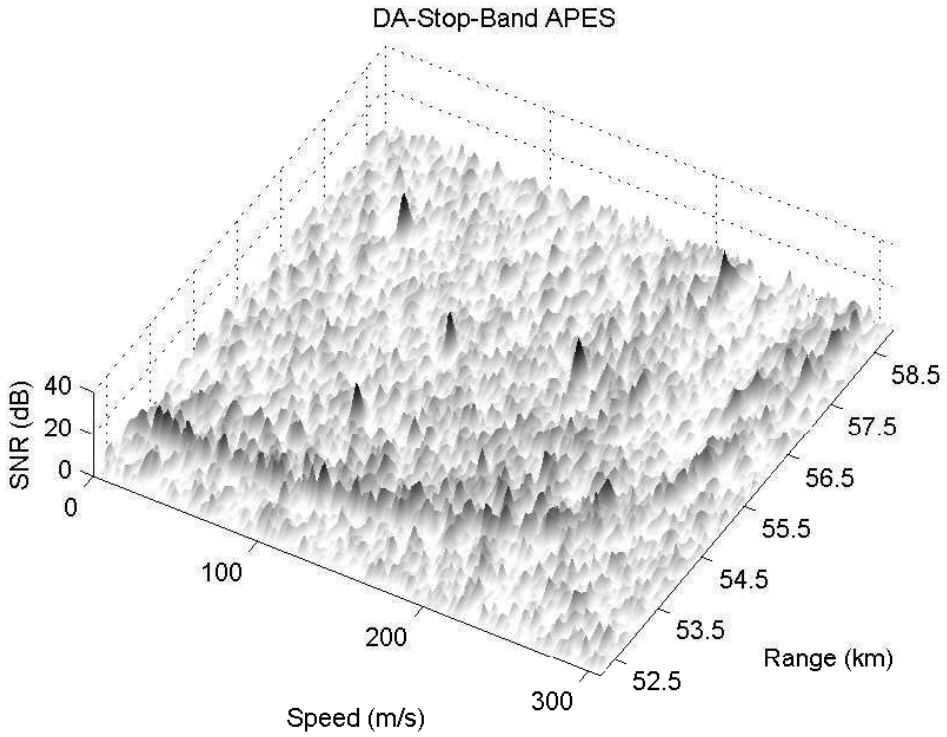


FIGURE 3.18 – Mode air-air : traitement Deterministic-Aided Stop-Band APES

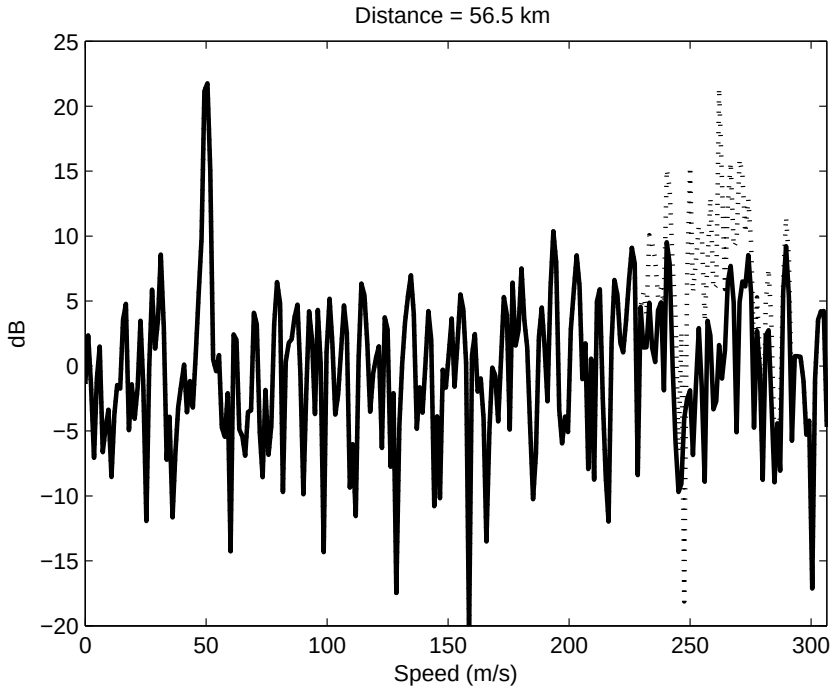


FIGURE 3.19 – Comparaison entre Stop-Band STAP (pointillés) et Deterministic-Aided Stop-Band (solide) sur les données air-air à une distance de 58.425 km

Le traitement Stop-Band supprime presque tout le fouillis des lobes secondaires mais n'arrive pas à correctement supprimer le fouillis du lobe principal (voir Fig. 3.17), alors que le traitement DA-Stop-Band (Fig. 3.18) supprime presque entièrement le fouillis. Sur la Figure 3.19, l'effet sur l'atténuation du fouillis du DA-Stop-Band est visible à travers une comparaison avec le traitement Stop-Band traditionnel. Nous pouvons observer que ces deux traitements Stop-Band ne suppriment pas complètement le fouillis des lobes secondaires. Ce problème peut-être résolu en utilisant des méthodes de sous-espace de type *Eigencanceller* et FAPI au lieu de l'inversion matricielle utilisée ici (cf. Chaptire 2).

3.5 Conclusion

Dans ce chapitre, nous avons proposé deux algorithmes utilisant du déterminisme pour améliorer les traitements basés sur l'algorithme APES. Le premier algorithme, qui repose sur la relation déterministe angle-Doppler du fouillis est particulièrement adapté à la détection GMTI. Les résultats sur données réelles montrent qu'il surpasse aussi bien le traitement STAP classique que les traitements basés sur la l'algorithme APES. Le second algorithme, dont le but est de supprimer la composante continue des interférences, est quant à lui bien adapté au mode air-air. Dans ce cas, la composante continue est le fouillis du lobe principal. Sur des données réalistes, il supprime totalement le fouillis du lobe principal alors que le traitement STAP classique et les méthodes basées sur APES échouent, causant un taux élevé de fausses alarmes.

4

Géométrie Riemannienne pour la sélection des données d'entrainement

Sommaire

Dans les chapitres précédents, nous avons considéré que l'environnement était très hétérogène, ceci étant dû à la non-stationnarité du fouillis ou à la trop forte densité de cibles. Bien sûr, aucun environnement n'est purement hétérogène ou purement homogène et il serait dommage de ne pas utiliser des données secondaires si elles sont homogènes aux données primaires. Il conviendrait d'utiliser une combinaison de données primaires et de données secondaires [56]. La sélection de ces données d'entrainement est cependant un point crucial dans l'utilisation de cette solution. Les méthodes classiques de sélection des données travaillent en géométrie Euclidienne et ne permettent pas de tenir pleinement compte des informations contenues dans la matrice de covariance. Nous étudions dans ce chapitre une nouvelle approche de la sélection des données secondaires en géométrie Riemannienne qui exploite le fait que les matrices de covariance sont définies positives.

4.1 Sélection des matrices de covariance d'entrainement

En traitement STAP, un des points cruciaux est la sélection des données d'entrainement. Les interférences contenues dans ces données doivent être homogènes aux interférences des données que nous voulons filtrer. Dans les chapitres précédents, nous avons vu plusieurs méthodes qui opèrent en n'utilisant pas de données provenant des données secondaires ce qui convient aux environnements fortement hétérogènes. En pratique, nous faisons face à une succession de zones plus ou moins homogènes ou hétérogènes. Il est donc important de savoir dans quelles situations il est possible d'utiliser des données secondaires pour estimer la matrice de covariance, et, le cas échéant, comment choisir ces données (cf Figure 4.1).

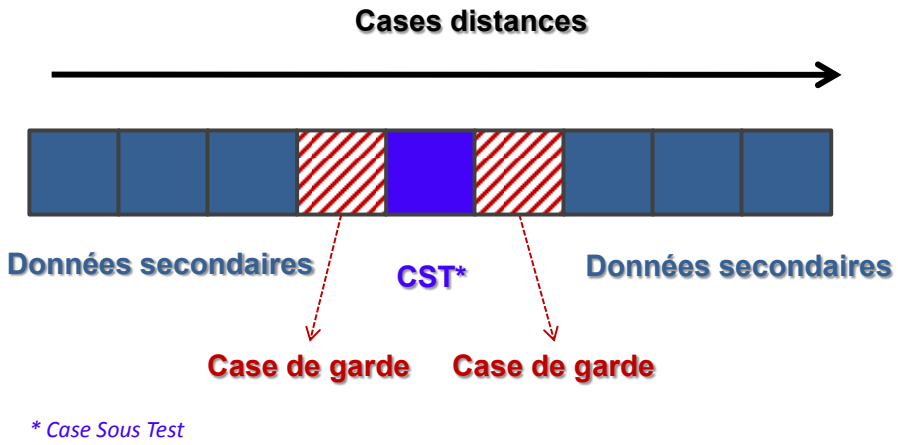


FIGURE 4.1 – Illustration de la sélection de données d'entrainement en STAP

4.1.1 Géométrie Euclidienne

Une des méthodes classiques pour sélectionner les données est d'utiliser un critère basé sur la puissance du signal contenu dans la matrice de covariance [21]. Ce critère de sélection basé sur la puissance est une application de la distance Euclidienne d_1 entre deux matrices, où l'on cherche le minimum de la norme de Frobenius de la matrice différence entre une matrice de covariance estimée sur les données secondaires $\mathbf{R}_l$ et la matrice de covariance de la case sous test $\mathbf{R}_0$:

$$d_1^2(\mathbf{R}_l, \mathbf{R}_0) = ||\mathbf{R}_l - \mathbf{R}_0||_F^2 = tr[(\mathbf{R}_l - \mathbf{R}_0)^H(\mathbf{R}_l - \mathbf{R}_0)] \tag{4.1}$$

L'équation (4.1) montre que cette méthode ne se base que sur la puissance du signal contenu dans les matrices de covariance. Elle ne tire pas profit de la structure de ces matrices. Ainsi deux matrices de covariance estimées sur des cases distances où sont présentes deux interférences de même puissance mais ayant des paramètres d'angle et de fréquence différents seront vues par ce critère comme très proches.

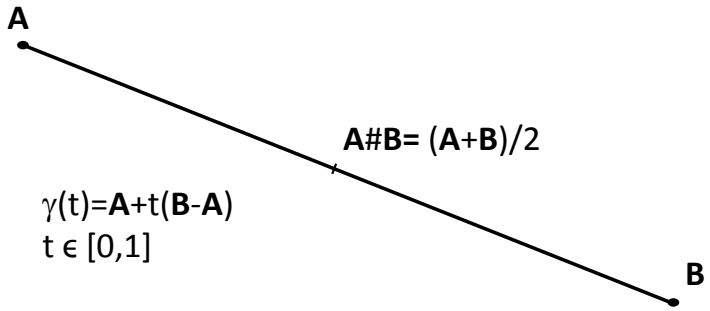


FIGURE 4.2 – Représentation de la moyenne arithmétique comme milieu de la géodésique $[\mathbf{A}, \mathbf{B}]$ sur un espace normé à courbure nulle

En géométrie Euclidienne, la géodésique dans l'espace normé est définie par (c.f Figure 4.2)

$$\gamma(t) = \mathbf{A} + t(\mathbf{B} - \mathbf{A}), \quad 0 \leq t \leq 1 \tag{4.2}$$

et nous retrouvons pour $t = 1/2$ la moyenne arithmétique classique que nous notons

$$\mathbf{A}\#_{1/2}\mathbf{B} = \frac{(\mathbf{A} + \mathbf{B})}{2} \tag{4.3}$$

et qui se généralise dans le cas de plusieurs matrices $\mathbf{A}_1, ..., \mathbf{A}_m$ par

$$\mathbf{M}_{\mathbf{A}}(\mathbf{A}_1, ..., \mathbf{A}_m) = \frac{1}{m}\sum_{i=1}^{m}\mathbf{A}_i \tag{4.4}$$

Il existe d'autres types de moyennes en géométrie Euclidienne comme la moyenne harmonique, contre-harmonique [57], *Root*-Euclidienne, *Power*-Euclidienne ou Log-Euclidienne.

- Log-Euclidienne :

$$d_{LE}(\mathbf{R}_l, \mathbf{R}_0) = \sqrt{tr[(\log \mathbf{R}_l - \log \mathbf{R}_0)^H(\log \mathbf{R}_l - \log \mathbf{R}_0)]} \tag{4.5}$$

- *Root*-Euclidienne :

$$d_{RE}(\mathbf{R}_l, \mathbf{R}_0) = \sqrt{tr[(\mathbf{R}_l{}^{1/2} - \mathbf{R}_0{}^{1/2})^H(\mathbf{R}_l{}^{1/2} - \mathbf{R}_0{}^{1/2})]} \tag{4.6}$$

- *Power*-Euclidienne :

$$d_{PE}(\mathbf{R}_l, \mathbf{R}_0) = \sqrt{tr[(\mathbf{R}_l{}^{\gamma} - \mathbf{R}_0{}^{\gamma})^H(\mathbf{R}_l{}^{\gamma} - \mathbf{R}_0{}^{\gamma})]} \tag{4.7}$$

Ces distances ont été testées pour une application STAP dans [58]. Dans la suite, nous allons nous intéresser à une autre approche basée sur la géométrie Riemannienne. Cette géométrie prend en compte le propriété définie positive des matrices de covariance.

4.2 Géométrie Riemannienne pour matrices définies positives

L'ensemble des matrices définies positives $n \times n$ sur $\mathbb{C}$ est une variété différentielle avec une structure Riemannienne [59] et forme un espace de Hilbert $\mathbb{M}_n$ avec $< \mathbf{A}, \mathbf{B} >= tr(\mathbf{A}^H\mathbf{B})$ comme produit scalaire et la norme associée, aussi appelée norme de Frobenius, $\| \mathbf{A} \|_F= tr(\mathbf{A}^H\mathbf{A})^{1/2}$. L'ensemble des matrices hermitiennes constitue un espace vectoriel $\mathbb{H}_n$ dans $\mathbb{M}_n$. Le sous-espace des matrices strictement positives $\mathbb{P}_n$ est un sous-espace ouvert dans $\mathbb{H}_n$, il s'agit donc d'une variété différentielle. L'espace tangent à $\mathbb{P}_n$ à un de ses points quelconque $\mathbf{A}$ est l'espace $T_{\mathbf{A}}\mathbb{P}_n = \{\mathbf{A}\} \times \mathbb{H}_n$. Le produit scalaire sur $\mathbb{H}_n$ mène à une métrique Riemannienne sur la variété $\mathbb{P}_n$.

4.2.1 Distance entre matrices

Cette métrique Riemannienne est déterminée au point $\mathbf{A}$ de la variété $\mathbb{P}_n$ par le différentiel :

$$ds =\| \mathbf{A}^{-1/2}\mathbf{dA}\mathbf{A}^{-1/2} \|_F= [tr(\mathbf{A}^{-1}\mathbf{dA})^2]^{1/2} \tag{4.8}$$

Si γ est un chemin différentiel dans $\mathbb{P}_n$ reliant a à b, sa longueur est définie par

$$L(\gamma) = \int_a^b \|\gamma^{-1/2}(t)\gamma'(t)\gamma^{-1/2}(t)\|dt \tag{4.9}$$

où $\gamma'(t)$ est la dérivée première de $\gamma(t)$. Quels que soient $\mathbf{A}$ et $\mathbf{B}$ deux points de $\mathbb{P}_n$, alors la distance $\delta(\mathbf{A}, \mathbf{B})$ entre ces points est définie par

$$\delta(\mathbf{A}, \mathbf{B}) = \inf\{L(\gamma) : \gamma \text{ est un chemin de } \mathbf{A} \text{ vers } \mathbf{B}\} \tag{4.10}$$

Théorème 1. *Soit* $\mathbf{A}$ *et* $\mathbf{B}$ *deux éléments de* $\mathbb{P}_n$. *Alors il existe une unique géodésique* $[\mathbf{A}, \mathbf{B}]$ *joignant* $\mathbf{A}$ *et* $\mathbf{B}$. *Cette géodésique a une paramétrisation*

$$\gamma(t) = \mathbf{A}^{-1/2}(\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2})^t\mathbf{A}^{-1/2}, \quad 0 \leq t \leq 1 \tag{4.11}$$

car, pour tout t *nous avons*

$$\delta(\mathbf{A}, \gamma(t)) = t\delta(\mathbf{A}, \mathbf{B}) \tag{4.12}$$

et de plus

$$\delta(\mathbf{A}, \mathbf{B}) =\| \log \mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2} \|_F \tag{4.13}$$

Démonstration. Les matrices $\mathbf{I}$ et $\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2}$ commutent. Donc la géodésique $[\mathbf{I}, \mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2}]$ est paramétrisée par

$$\gamma_0(t) = (\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2})^t \tag{4.14}$$

En appliquant l'isométrie $\Gamma_{\mathbf{A}^{-1/2}}$, nous obtenons le chemin

$$\gamma(t) = \Gamma_{\mathbf{A}^{-1/2}}(\gamma_0(t)) = \mathbf{A}^{-1/2}(\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2})^t\mathbf{A}^{-1/2} \tag{4.15}$$

Ce chemin joint les points $\Gamma_{\mathbf{A}^{-1/2}}(\mathbf{I}) = \mathbf{A}$ et $\Gamma_{\mathbf{A}^{-1/2}}(\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2}) = \mathbf{B}$.
Comme $\Gamma_{\mathbf{A}^{-1/2}}$ est une isométrie (voir Annexe B.6.1), ce chemin est la géodésique $[\mathbf{A}, \mathbf{B}]$ et

$$\delta(\mathbf{A}, \mathbf{B}) = \delta(\mathbf{I}, \mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2}) \tag{4.16}$$

En utilisant la Proposition 1 de l'Annexe B.6.2, nous voyons que (4.16) s'écrit

$$\begin{aligned} \delta(\mathbf{A}, \mathbf{B}) =& \| \log \mathbf{I} - \log(\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2}) \|_F \\ =& \| \log \mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2} \|_F \end{aligned} \tag{4.17}$$

□

La formule (4.13) donne une représentation formelle de la métrique δ définie en (4.9). Il s'agit de la métrique Riemannienne sur la variété $\mathbb{P}_n$. D'après la définition de la norme de Frobenius, nous voyons que

$$\delta(\mathbf{A}, \mathbf{B}) = (\sum_{i=1}^{n} \log^2 \lambda_i(\mathbf{A}^{-1}\mathbf{B}))^{1/2} \tag{4.18}$$

où λ_i sont les valeurs propres de la matrice $\mathbf{A}^{-1}\mathbf{B}$.

Nous appellerons distance Riemannienne entre deux matrices de covariance définies positives $\mathbf{R}_1$ et $\mathbf{R}_2$ suivant (4.13) et (4.18), la distance

$$d_3^2(\mathbf{R_1}, \mathbf{R_2}) = || \log(\mathbf{R_1^{-1/2}}\mathbf{R_2}\mathbf{R_1^{-1/2}})||_F^2 = \sum_{i=1}^{N} \log^2(\lambda_i) \tag{4.19}$$

où λ_i sont les valeurs propres de la matrice $\mathbf{R_1^{-1}R_2}$ c'est à dire les solutions de $\det(\mathbf{R_2} - \lambda\mathbf{R_1}) = 0$.

4.2.2 Moyenne géométrique de matrices définies positives

Nous avons vu dans la Section 4.1.1 qu'en géométrie Euclidienne, la moyenne de deux matrices est définie comme étant le milieu de la géodésique $\gamma(t) = \mathbf{A} + t(\mathbf{B} - \mathbf{A})$ ce qui revient à la moyenne arithmétique des équations (4.3) et (4.4). La moyenne harmonique est aussi bien connue et est définie par

$$\mathbf{M_{harm}} = \left(\frac{\mathbf{A^{-1}} + \mathbf{B^{-1}}}{2} \right)^{-1}$$

En revanche, la moyenne géométrique $\mathbf{M_{geo}}$, définie pour deux nombres réels a et b comme étant $\sqrt{ab}$ n'est pas définie pour les matrices $\mathbf{A}$ et $\mathbf{B}$

$$\mathbf{M_{geo}} \neq \mathbf{A^{1/2}B^{1/2}} \tag{4.20}$$

En effet, la multiplication de matrices n'étant pas commutative, $\mathbf{A^{1/2}B^{1/2}} \neq \mathbf{B^{1/2}A^{1/2}}$.

En utilisant les résultats précédents, la moyenne géométrique de deux matrices définies positives $\mathbf{A}$ et $\mathbf{B}$, notée $\mathbf{A}\#_{1/2}\mathbf{B}$ peut-être définie [60] comme étant le milieu de la géodésique γ (voir Eq. (4.15)) qui relie $\mathbf{A}$ et $\mathbf{B}$ dans l'espace $(\mathbb{P}_n, \delta)$, c'est à dire pour $\gamma(t = 1/2)$:

$$\mathbf{A}\#_{1/2}\mathbf{B} = \mathbf{A}^{\mathbf{1/2}}(\mathbf{A}^{\mathbf{-1/2}}\mathbf{B}\mathbf{A}^{\mathbf{-1/2}})^{1/2}\mathbf{A}^{\mathbf{1/2}} \tag{4.21}$$

D'après l'équation (4.11), il est clair que le milieu de la géodésique $[\mathbf{A}, \mathbf{B}]$ est le même que le milieu de la géodésique $[\mathbf{B}, \mathbf{A}]$ (c.f Figure 4.3).

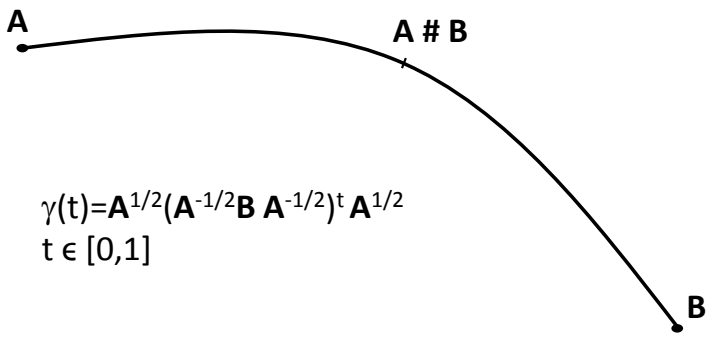


FIGURE 4.3 – Représentation de la moyenne géométrique comme milieu de la géodésique $[\mathbf{A}, \mathbf{B}]$ sur un espace métrique à courbure négative

Proposition 1. *Soit* $\mathbf{A}$*,* $\mathbf{B}$ *et* $\mathbf{C}$ *trois points de* $\mathbb{P}_n$*. Alors*

$$\delta(\mathbf{A}\#\mathbf{B}, \mathbf{A}\#\mathbf{C}) \leq \frac{\delta(\mathbf{B}, \mathbf{C})}{2} \tag{4.22}$$

Démonstration (voir Annexe B.6.3), illustration sur la Figure 4.4. □

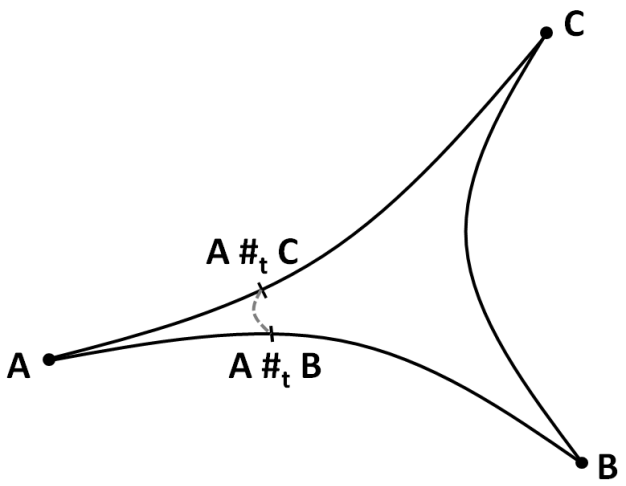


FIGURE 4.4 – Illustration de la convexité de la métrique à travers la Proposition 1.

La proposition démontre la convexité de la métrique, c'est à dire que la métrique est à courbure négative. Il est facile de démontrer qu'en géométrie Euclidienne, l'inégalité (4.22) devient une égalité, la courbure de la métrique est alors nulle.

La moyenne géométrique de $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ doit se trouver au centre du triangle de sommets $\mathbf{A}$, $\mathbf{B}$ et $\mathbf{C}$. Plus généralement, pour $\mathbf{A_1}$,..., $\mathbf{A_m}$ dans $\mathbb{P}_n$, la moyenne géométrique $\mathbf{M_{geo}}(\mathbf{A_1}, ..., \mathbf{A_m})$ est le barycentre $\mathbf{G}$ de l'enveloppe convexe de $\mathbf{A_1}$,..., $\mathbf{A_m}$.

$$\mathbf{G}(\mathbf{A_1}, ..., \mathbf{A_m}) = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{m} \delta^2(\mathbf{X}, \mathbf{A_i}) \tag{4.23}$$

Un minimum existe et est unique [61]. Différentes méthodes numériques ont été développées afin de calculer cette moyenne géométrique efficacement [62][63].

4.2.3 Autres développements

Il est intéressant de remarquer que la métrique définie dans (4.8) est un cas particulier de la géométrie des espaces de Siegel [64] définis par

$$SH_n = \{\mathbf{Z} = \mathbf{X} + i\mathbf{Y} \in Sym(n, \mathbb{C}) / Im(\mathbf{Z}) = \mathbf{Y} > 0\}$$

$$ds^2_{Siegel} = [tr(\mathbf{Y}^{-1}(\mathbf{dZ})\mathbf{Y}^{-1}(\mathbf{dZ}^*))]^{1/2} \tag{4.24}$$

La géométrie différentielle des matrices hermitiennes définies positives est un cas particulier de cette géométrie [65] où l'on se restreint au demi-plan imaginaire, i.e $\mathbf{X} = 0$ et $\mathbf{Y} = i\mathbf{A}$ ce qui donne la métrique vue précédemment

$$ds^2 = [tr(\mathbf{A}^{-1}\mathbf{dA})^2] \tag{4.25}$$

Il est possible de définir un flot de gradients sur cette variété, appelé flot de Karcher [66], permettant de calculer de manière itérative le barycentre de N matrices de covariance vu dans (4.23)

$$\mathbf{G_k} = (\mathbf{G_{k-1}^{-1/2}})^H \exp\left(\epsilon \sum_{i=1}^{K} \log((\mathbf{G_{k-1}^{-1/2}})^H \mathbf{G_i} \mathbf{G_{k-1}^{-1/2}})\right) \mathbf{G_{k-1}^{-1/2}}$$

Dans [67], les auteurs utilisent cette méthode pour estimer la matrice de covariance qui convergent ainsi vers le barycentre de Karcher. Le flot de Karcher ne conserve cependant pas la structure Toeplitz d'une matrice. Pour conserver cette propriété, il convient d'utiliser une paramétrisation auto-régressive complexe (théorèmes de Trench et Verblunsky) [68][69]. Il en est de même pour les matrices Toeplitz block-Toeplitz hermitiennes définies positives [65].

Dans la suite, nous allons proposer un nouveau critère pour la sélection des données en vue de l'estimation de la matrice de covariance et nous montrerons qu'il peut être vu à travers une approche de la géométrie Riemannienne.

4.3 Recherche du critère optimal pour la sélection des données

4.3.1 Approche physique

Notre approche de ce problème est de prendre un point de vue physique pour trouver un critère, qui pourra être abusivement appelé "distance" par comparaison avec les distances Euclidienne et Riemannienne, qui satisfait le problème de minimisation du filtre STAP.

Soit $\mathbf{w_l^H} = \mathbf{s_s^H R_l}^{-1}$ les poids du filtre obtenus avec la matrice de covariance $\mathbf{R}_l$ qui est estimée sur une case distance voisine l. Nous appliquons ce filtre à un vecteur de données de la case $\mathbf{x}_0$:

$$\begin{aligned} \mathbf{z} &= \mathbf{w_l^H x}_0 \\ \mathbf{z} &= \mathbf{s}_s^H \mathbf{R_l}^{-1/2} \mathbf{R_l}^{-1/2} \mathbf{x}_0 \\ \mathbf{z} &= \mathbf{s_s^H R_l}^{-1/2} \mathbf{y} \end{aligned} \tag{4.26}$$

Le terme $\mathbf{s_s^H R_l}^{-1/2}$ représente le filtre adapté alors que le terme $\mathbf{y} = \mathbf{R_l}^{-1/2}\mathbf{x}_0$ est le signal blanchi si $\mathbf{w_l^H}$ est le filtre optimal. Nous cherchons justement la matrice $\mathbf{R}_l$ telle que $\mathbf{w_l^H}$ soit le filtre optimal, c'est à dire que le signal $\mathbf{y}$ soit blanc. En conséquence, nous voulons que $E\{\mathbf{yy^H}\} = \mathbf{I}$, i.e :

$$E\{\mathbf{R_l}^{-1/2}\mathbf{x}_0\mathbf{x}_0^\mathbf{H}\mathbf{R_l}^{-1/2}\} = \mathbf{I} \tag{4.27}$$

$$\mathbf{R_l}^{-1/2} E\{\mathbf{x}_0\mathbf{x}_0^\mathbf{H}\}\mathbf{R_l}^{-1/2} = \mathbf{I} \tag{4.28}$$

$$\mathbf{R_l}^{-1/2}\mathbf{R}_0\mathbf{R_l}^{-1/2} - \mathbf{I} = 0 \tag{4.29}$$

Ce qui mène à la "distance" physique que nous voulons minimiser :

$$d_2^2(\mathbf{R}_l, \mathbf{R}_0) = ||\mathbf{R}_l^{-1/2}\mathbf{R}_0\mathbf{R}_l^{-1/2} - \mathbf{I}||_F^2 \tag{4.30}$$

Pour filtrer les données $\mathbf{x}_0$ dont $\mathbf{R}_0$ est la matrice de covariance, nous recherchons donc les matrices de covariance adjacentes $\mathbf{R}_l$ dont les distances d_2^2 à $\mathbf{R}_0$ sont minimales.
Il est possible de simplifier d_2. En effet

$$\begin{aligned} d_2(\mathbf{R}_l, \mathbf{R}_0) =& \| \mathbf{R}_l^{-1/2}\mathbf{R}_0\mathbf{R}_l^{-1/2} - \mathbf{I} \|_F \\ &= tr[(\mathbf{R}_l^{-1/2}\mathbf{R}_0\mathbf{R}_l^{-1/2} - \mathbf{I})^\mathbf{H}(\mathbf{R}_l^{-1/2}\mathbf{R}_0\mathbf{R}_l^{-1/2} - \mathbf{I})]. \\ &= tr[(\mathbf{R}_l^{-1/2})^\mathbf{H}\mathbf{R}_0{}^\mathbf{H}(\mathbf{R}_l^{-1/2})^\mathbf{H}\mathbf{R}_l^{-1/2}\mathbf{R}_0\mathbf{R}_l^{-1/2} - 2.\mathbf{R}_l^{-1/2}\mathbf{R}_0\mathbf{R}_l^{-1/2} + \mathbf{I}]. \\ &= tr[\mathbf{R}_l^{-1}\mathbf{R}_0\mathbf{R}_l^{-1}\mathbf{R}_0 - 2.\mathbf{R}_l^{-1}\mathbf{R}_0 + \mathbf{I}]. \\ &= tr[(\mathbf{R}_l^{-1}\mathbf{R}_0 - \mathbf{I})^2]. \\ &= tr[(\mathbf{R}_l^{-1}\mathbf{R}_0 - \mathbf{I})^\mathbf{H}(\mathbf{R}_l^{-1}\mathbf{R}_0 - \mathbf{I})]. \end{aligned}$$

Il vient une forme plus simple et plus facile à calculer que la distance Riemannienne

$$d_2^2(\mathbf{R}_l, \mathbf{R}_0) = ||\mathbf{R}_l^{-1}\mathbf{R}_0 - \mathbf{I}||_F^2 \tag{4.31}$$

Le critère décrit dans (4.31) appelé "distance physique" ne satisfait pas toutes les propriétés d'une distance mathématique. En particulier, d_2 n'est pas symétrique :

$$d_2(\mathbf{R}_l, \mathbf{R}_0) \neq d_2(\mathbf{R}_0, \mathbf{R}_l) \tag{4.32}$$

En particulier, si nous mesurons la distance entre deux matrices de covariance telles que $\mathbf{R}_l = \alpha\mathbf{R_0}$, nous avons

$$\begin{aligned} d_2(\mathbf{R}_l, \mathbf{R}_0) =& \| \mathbf{R}_l^{-1}\mathbf{R}_0 - \mathbf{I} \|_F \\ =& \| (\alpha\mathbf{R}_0)^{-1}\mathbf{R}_0 - \mathbf{I} \|_F \\ =& \| (\frac{1}{\alpha} - 1)\mathbf{I} \|_F \\ =& |\frac{1}{\alpha} - 1| \, \| \mathbf{I} \|_F \\ =& |\frac{1}{\alpha} - 1|\sqrt{NM} \end{aligned}$$

Lorsque $\alpha \ll 1$, la distance augmentera, ce qui est normal d'un point de vue physique puisque le signal ne sera plus blanchi et la réjection des interférences sera mauvaise. En revanche, $\alpha \gg 1$, la distance n'augmentera pas puisque $(\frac{1}{\alpha} - 1) \longrightarrow 1$. Cette expression fait apparaitre une normalisation naturelle pour cette distance qui est calculée en divisant cette dernière par $\| \mathbf{I} \|_F = \sqrt{NM}$.

Nous pouvons résoudre ce problème de non-symétrie en définissant une "distance" physique symétrique :

$$d_4^2(\mathbf{R}_l, \mathbf{R}_0) = [\frac{1}{2}(d_2(\mathbf{R_l}, \mathbf{R_0}) + d_2(\mathbf{R_0}, \mathbf{R_l}))]^2 \tag{4.33}$$

Dans ce cas, si $\mathbf{R}_l = \alpha\mathbf{R_0}$ alors

$$\begin{aligned} d_4(\mathbf{R}_l, \mathbf{R}_0) &= \frac{1}{2}\left(\| \mathbf{R}_l^{-1}\mathbf{R}_0 - \mathbf{I} \|_F + \| \mathbf{R}_0^{-1}\mathbf{R}_l - \mathbf{I} \|_F \right) \\ &= \frac{1}{2}\left(\| \frac{1}{\alpha}\mathbf{R}_0{}^{-1}\mathbf{R}_0 - \mathbf{I} \|_F + \| \alpha\mathbf{R}_0{}^{-1}\mathbf{R}_0 - \mathbf{I} \|_F \right) \\ &= \frac{1}{2}\left(\| (\frac{1}{\alpha} - 1)\mathbf{I} \|_F + \| (\alpha - 1)\mathbf{I} \|_F \right) \\ &= \frac{1}{2}\left[(|\frac{1}{\alpha} - 1| + |\alpha - 1|)\sqrt{NM} \right] \end{aligned}$$

Si $\alpha \ll 1$, le terme $|\alpha - 1| \approx 1$ alors que le terme $|\frac{1}{\alpha} - 1| \gg 1$ donc la distance augmentera. De même si $\alpha \gg 1$, le terme $|\alpha - 1| \gg 1$ et $|\frac{1}{\alpha} - 1| \approx 1$. La distance augmentera de la même façon dans les deux cas et $d_2(\mathbf{R}_l, \mathbf{R}_0) = d_2(\mathbf{R}_0, \mathbf{R}_l)$.

4.3.2 Lien avec la géométrie Riemannienne

Comme nous l'avons vu dans la Section 4.2, lorsque nous travaillons sur des matrices hermitiennes définies positives, il est souhaitable de travailler sur la métrique Riemannienne. La distance entre deux matrices $\mathbf{R}_1$ et $\mathbf{R}_2$ est alors définie par

$$d_3^2(\mathbf{R_1},\mathbf{R_2}) = \| \log(\mathbf{R_1^{-1/2}R_2R_1^{-1/2}}) \|_F^2 = \sum_{k=1}^{N} \log^2(\lambda_k) \tag{4.34}$$

où λ_k sont les valeurs propres de la matrice $\mathbf{R_1^{-1}R_2}$ c'est à dire les solutions de $\det(\mathbf{R_2} - \lambda\mathbf{R_1}) = 0$.

Comme nous l'avons vu dans (4.30), le terme $\mathbf{R_1^{-1/2}R_2R_1^{-1/2}}$ est proche de la matrice identité $\mathbf{I}$. En conséquence, nous pouvons approximer (4.34) au premier ordre du développement en série de Taylor suivant

$$\log(\mathbf{A}) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{(\mathbf{A}-\mathbf{I})^n}{n} \tag{4.35}$$

En remplaçant $\mathbf{A}$ par $\mathbf{R_1^{-1/2}R_2R_1^{-1/2}}$ dans (4.35) nous avons

$$\log(\mathbf{R_1^{-1/2}R_2R_1^{-1/2}}) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{(\mathbf{R_1^{-1/2}R_2R_1^{-1/2}}-\mathbf{I})^n}{n} \tag{4.36}$$

Cette dernière équation nous donne, pour un développement à l'ordre 1, l'approximation suivante

$$\log(\mathbf{R_1^{-1/2}R_2R_1^{-1/2}}) \approx \mathbf{R_1^{-1/2}R_2R_1^{-1/2}} - \mathbf{I} \tag{4.37}$$

Nous pouvons alors approximer la distance Riemannienne de l'Eq. (4.34) par

$$d_3^2(\mathbf{R_1},\mathbf{R_2}) \approx ||\mathbf{R_1^{-1/2}R_2R_1^{-1/2}} - \mathbf{I}||_F^2 \tag{4.38}$$

Nous voyons que l'approximation de la distance Riemannienne, définie dans la Section 4.2.1, correspond exactement au critère "distance physique" de l'équation (4.30) proposée dans l'équation (4.34) de la Section précédente. Ce critère semble donc être bien meilleur pour la sélection de données d'entrainement que la distance Euclidienne (voir Section 4.1.1).

Remarquons que la distance métrique en géométrie Riemannienne (Eq. (4.34)) est en fait la distance Log-Euclidienne de l'équation (4.5). En effet,

$$d_3(\mathbf{R_1},\mathbf{R_2}) = \| \log(\mathbf{R_1^{-1/2}R_2R_1^{-1/2}}) \|_F = \| \log(\mathbf{R_2}) - \log(\mathbf{R_1}) \|_F$$

Cela est aussi clairement montré par la **Proposition 1.** de l'annexe B.6.2.

4.4 Résultats

4.4.1 Détection d'hétérogénéités sur données synthétiques Air-Sol

Les performances des différentes distances décrites précédemment sont comparées sur les données synthétiques réalistes **ONERA-AS** (voir Annexe A.3.1). La dimension temporelle du vecteur STAP est fixée à 6 retards ($M = 6$), ce qui implique que chaque matrice est estimée sur $M_p = 59$ échantillons. Un *diagonal loading* d'un facteur $\delta = 0.1$ est ajouté à chaque matrice.

Dans ces données, différents types de fouillis à différentes cases distances sont présents. La zone **A** est composée de fouillis arrivant par les lobes secondaires. Dans les zone **B** et **D** le fouillis est Gaussien et de puissance σ_0 et $\sigma_0/10$ respectivement. La zone **C** contient du fouillis piqué. Les zones **A** et **C** sont très hétérogènes, c'est à dire qu'il n'est pas possible d'utiliser de cases distance adjacentes pour estimer les poids du filtre alors que les zones **B** and **D** contiennent un fouillis homogène, et il est possible d'utiliser un grand nombre de cases distance tant qu'elles sont dans ces intervalles.

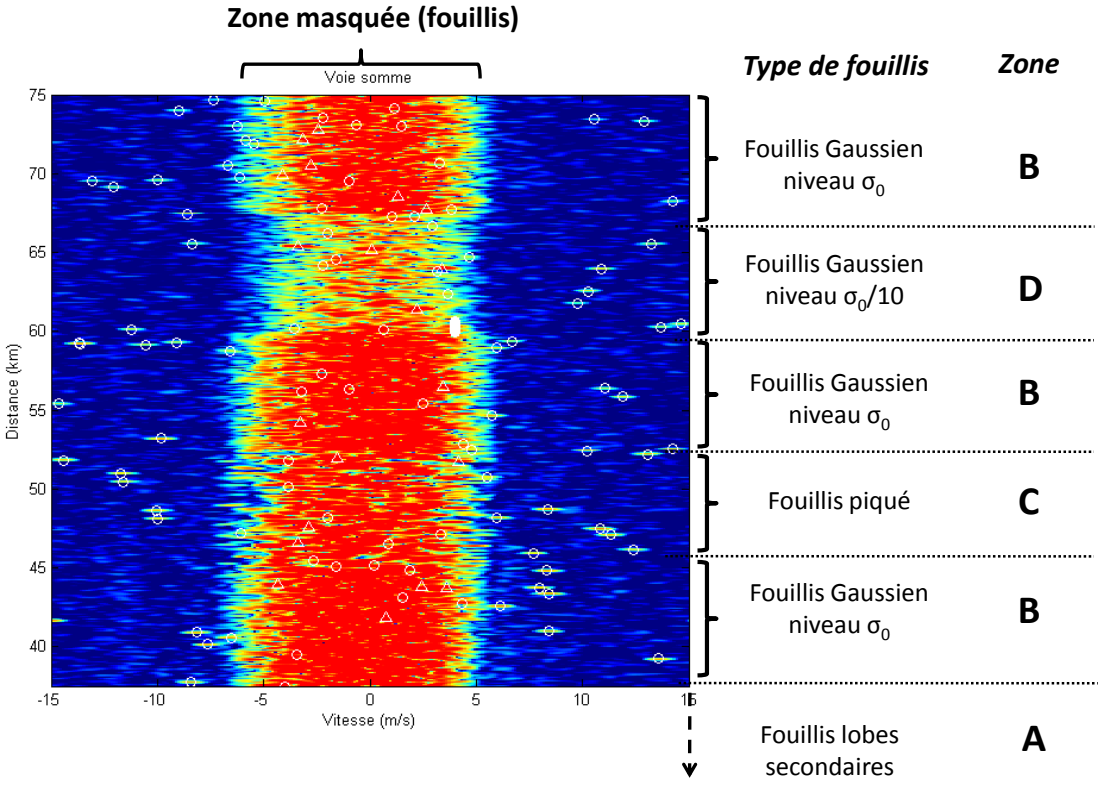


FIGURE 4.5 – Image vitesse-distance avec les différents type de fouillis

Nous affichons, pour chaque case distance, la distance normalisée entre la matrice de covariance de la case sous-test $l = 0$ et la matrice de covariance de la case distance $l = 10$. La normalisation est obtenue en divisant la distance par la distance moyenne entre deux matrices de covariance de même taille et estimées sur le même nombre d'échantillons qui ne contiennent que du bruit.

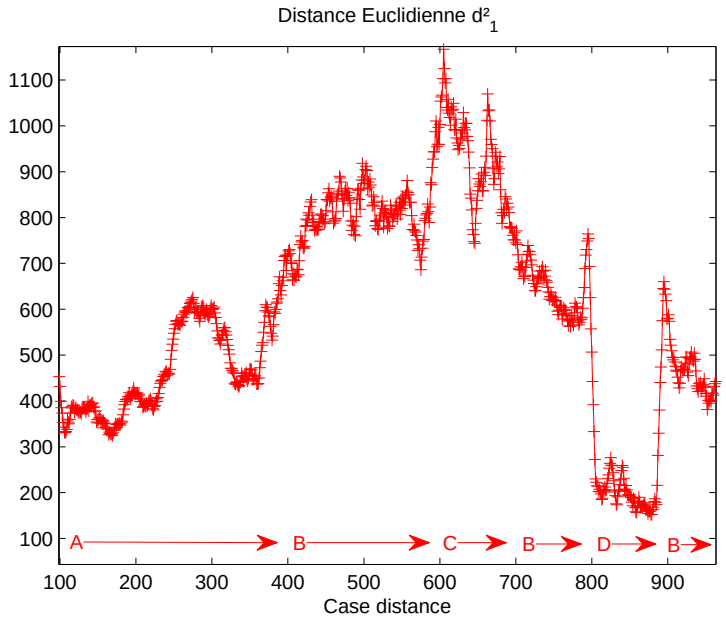


FIGURE 4.6 – Distance Euclidienne entre $\mathbf{R}_0$ et $\mathbf{R}_{10}$ le long de l'axe distance

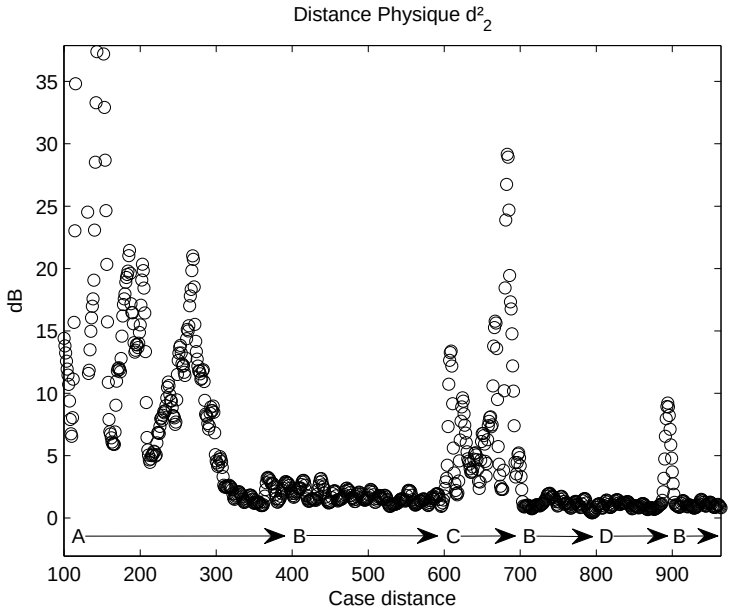


FIGURE 4.7 – Distance "physique" entre $\mathbf{R}_0$ et $\mathbf{R}_{10}$ le long de l'axe distance

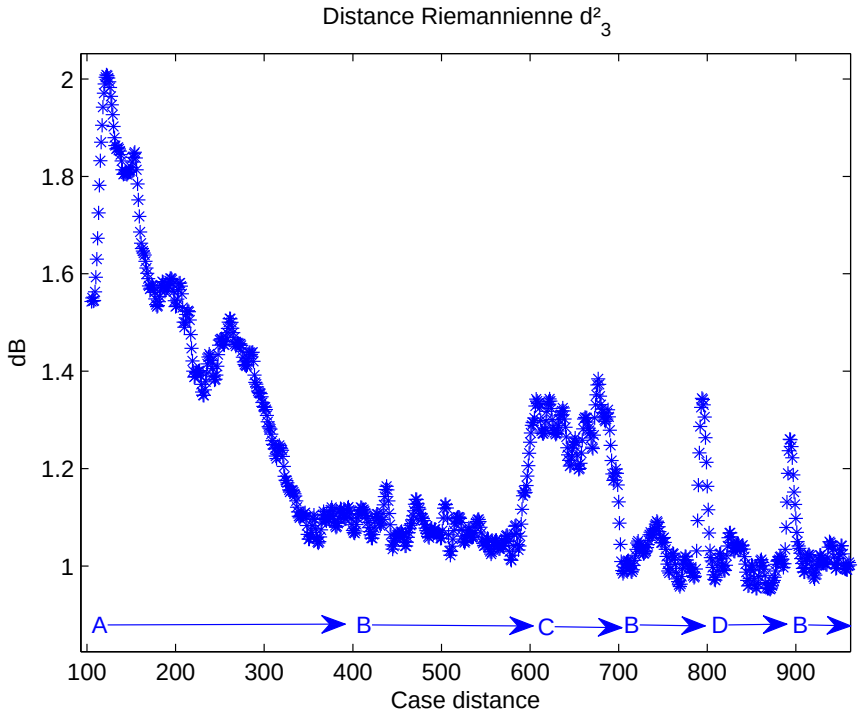


FIGURE 4.8 – Distance Riemannienne entre $\mathbf{R}_0$ et $\mathbf{R}_{10}$ le long de l'axe distance

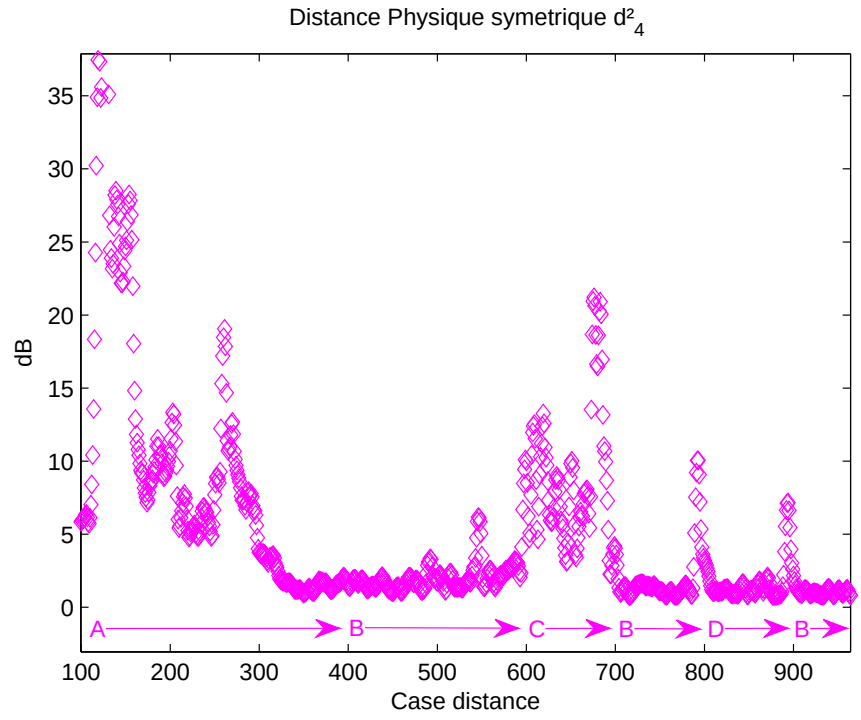


FIGURE 4.9 – Distance "physique" symétrique entre $\mathbf{R}_0$ et $\mathbf{R}_{10}$ le long de l'axe distance

Les résultats de la distance Euclidienne Fig. 4.6 montrent qu'il n'est pas possible de distinguer les différents types de fouillis le long de l'axe distance, sauf, comme prévu, pour la zone **D** qui est une zone de fouillis Gaussien moins puissant. Sur la Fig. 4.7 cependant, la "distance physique" permet de séparer presque toutes les zones de fouillis. Elle échoue seulement à détecter la transition entre le fouillis de puissance σ_0 et le fouillis de puissance $\sigma_{0/10}$. Enfin, sur la 4.8, la distance Riemannienne se comporte très bien puisqu'elle permet de détecter tous les changements de type de fouillis. Sur les Fig. 4.7 et Fig. 4.8, les distances entre matrices mettent en évidence les zones hétérogènes où il n'est pas possible d'utiliser de données secondaires. Sur la Fig. 4.9, comme prévu, la distance symétrique "physique" parvient à détecter le changement de fouillis à la case distance 800 aussi bien que d_3. En dehors de cela, les distances d_4 et d_2 sont très proches.

4.4.2 Détection d'hétérogénéités sur données synthétiques Air-Air

Nous testons maintenant ces distances sur les données **ONERA-AA**. Ici, le fouillis arrivant par le lobe principal est homogène en distance, comme nous pouvons l'observer sur la Figure 4.10 (raie centrée sur la vitesse 260 m/s). En revanche, le fouillis des lobes secondaires est très présent des cases distance 20 à 40 et est fortement hétérogène en distance. Nous fixons 8 retards ($M = 8$) pour le vecteur STAP, ce qui implique que chaque matrice est estimée sur $M_p = 121$ échantillons.

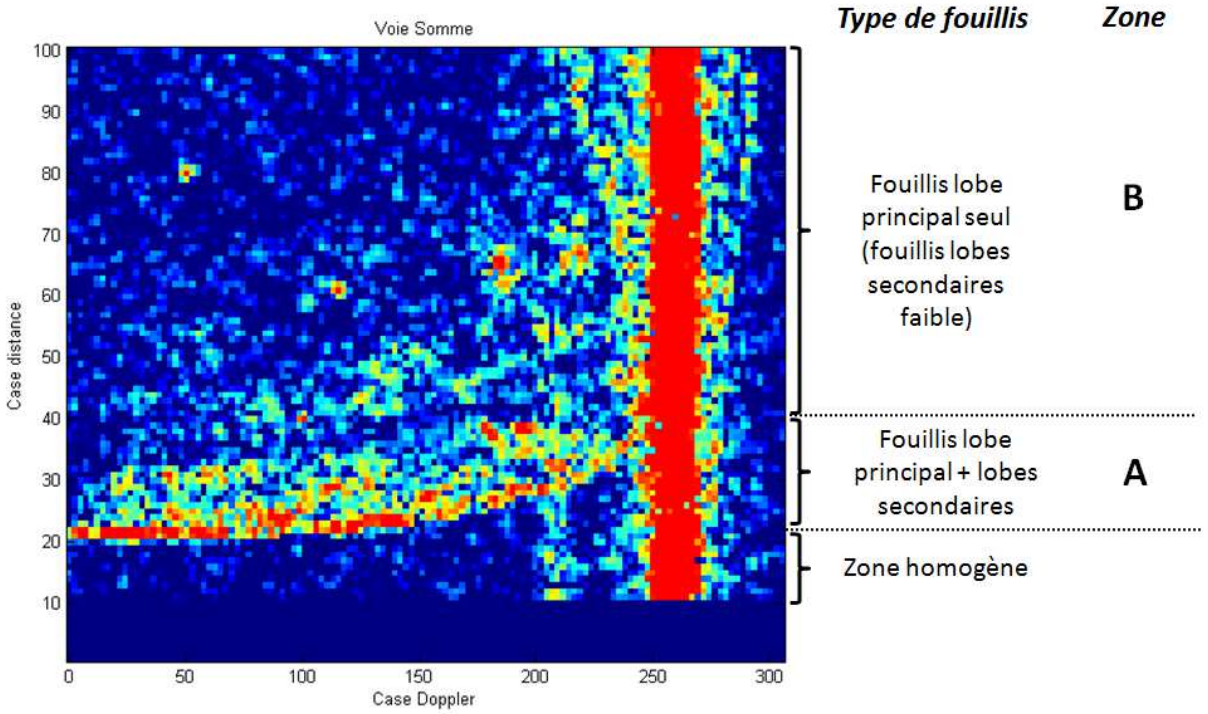


FIGURE 4.10 – Image vitesse-distance de la voie somme des données ONERA Air-Air

Nous affichons, pour chaque case distance, la distance normalisée entre la matrice de covariance de la case sous-test $l = 0$ et la matrice de covariance de la case distance $l = 5$. La normalisation est obtenue en divisant la distance par la distance moyenne entre deux matrices de covariance de même taille et estimées sur le même nombre d'échantillons qui ne contiennent que du bruit.

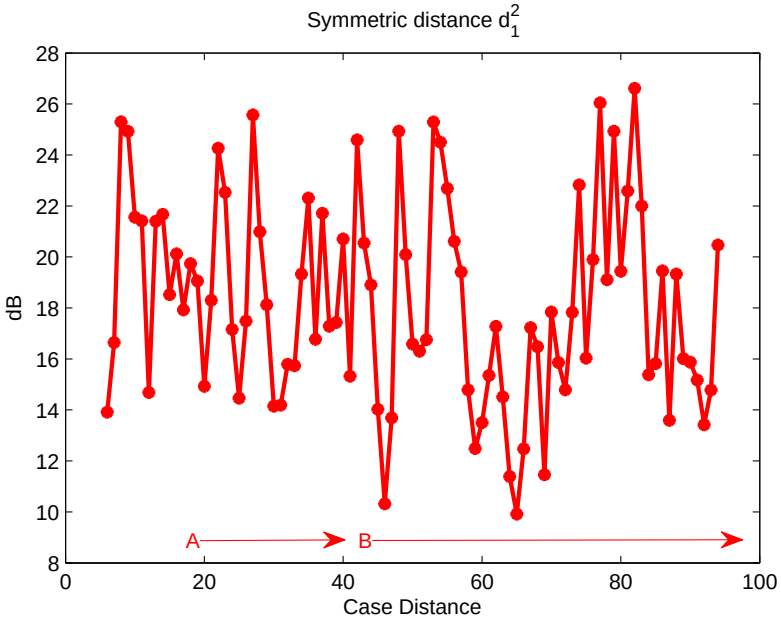


FIGURE 4.11 – Distance Euclidienne entre $\mathbf{R}_0$ et $\mathbf{R}_5$ le long de l'axe distance

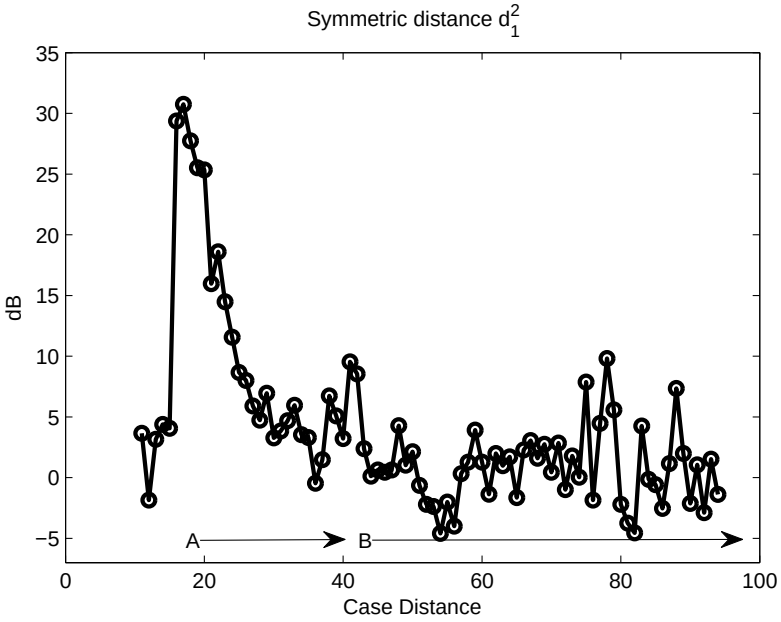


FIGURE 4.12 – Distance "physique" entre $\mathbf{R}_0$ et $\mathbf{R}_5$ le long de l'axe distance

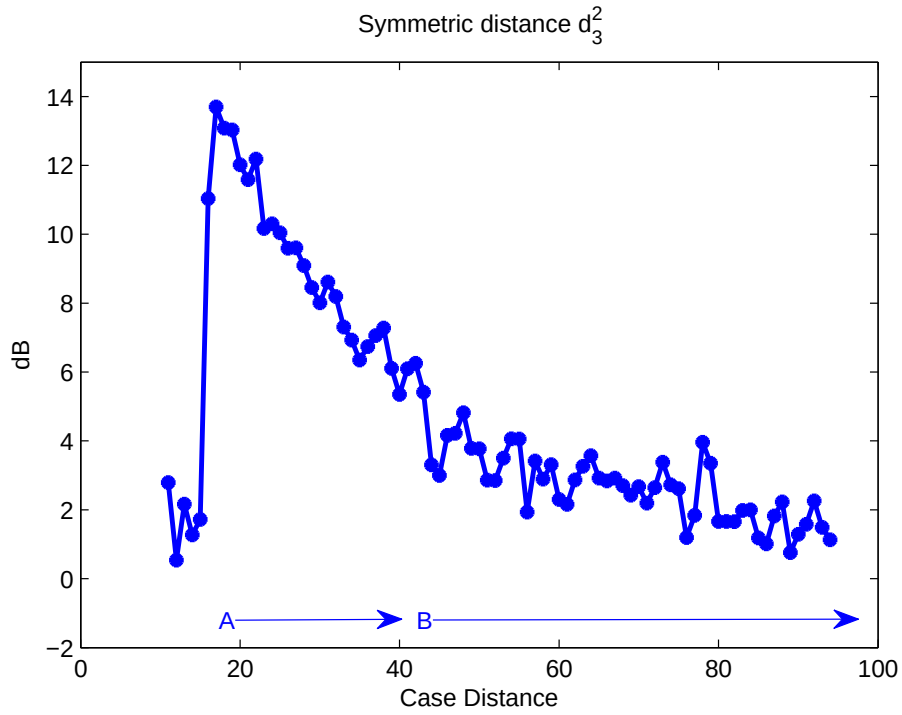


FIGURE 4.13 – Distance Riemannienne entre $\mathbf{R}_0$ et $\mathbf{R}_5$ le long de l'axe distance

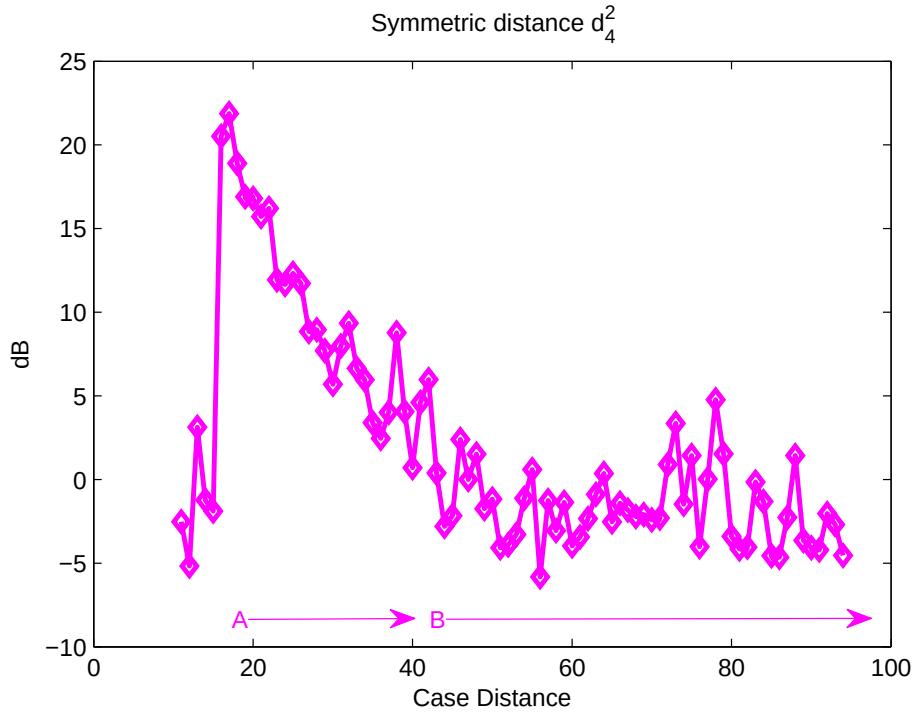


FIGURE 4.14 – Distance "physique" symétrique entre $\mathbf{R}_0$ et $\mathbf{R}_5$ le long de l'axe distance

Les résultats des figures 4.11 à 4.14 montrent que, comme dans le cas précédent, la distance Riemannienne et les deux critères physiques permettent de séparer la zone **A** où le fouillis hétérogène des lobes secondaires est très présent de la zone **B** où l'énergie des lobes secondaires n'est plus significative et qui devient donc homogène. Nous observons aussi la décroissance de cette énergie dans la zone **A** ainsi qu'une zone homogène sur les quelques cases distance qui précèdent la zone **A**. La distance Euclidienne ne permet pas de différencier les deux zones.

4.4.3 Traitement STAP sur données réelles

Nous comparons deux traitements STAP classiques sur les données réelles **SAREX** (voir Annexe A.3.4), l'un avec une sélection des données d'entrainement en géométrie Euclidienne et le second en géométrie Riemannienne. La dimension temporelle du vecteur STAP est fixée à 7 retards ($M = 7$), ce qui implique que chaque matrice est estimée sur $M_p = 58$ échantillons. Trois cibles sont présentes dans la scène (voir Fig 4.15, entourés) et les images Doppler distance après traitement STAP (Fig 4.16 et 4.17) sont seuillées à 10 dB par rapport au niveau de bruit moyen.

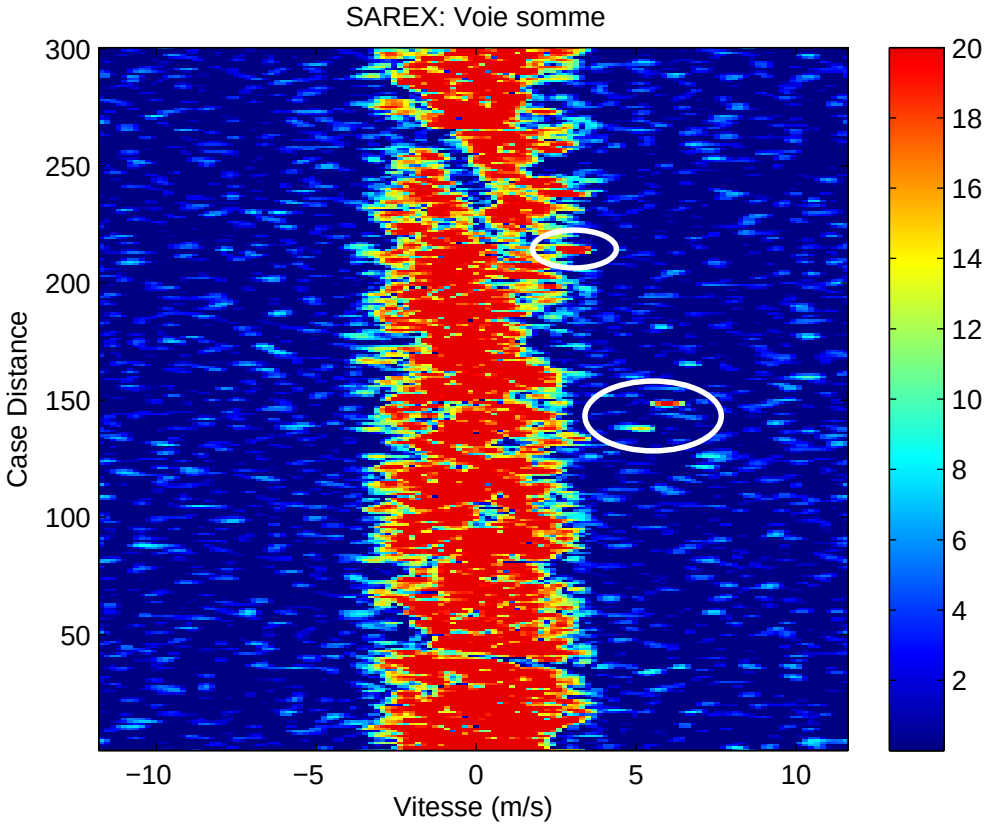


FIGURE 4.15 – Image vitesse-distance de la voie somme des données SAREX

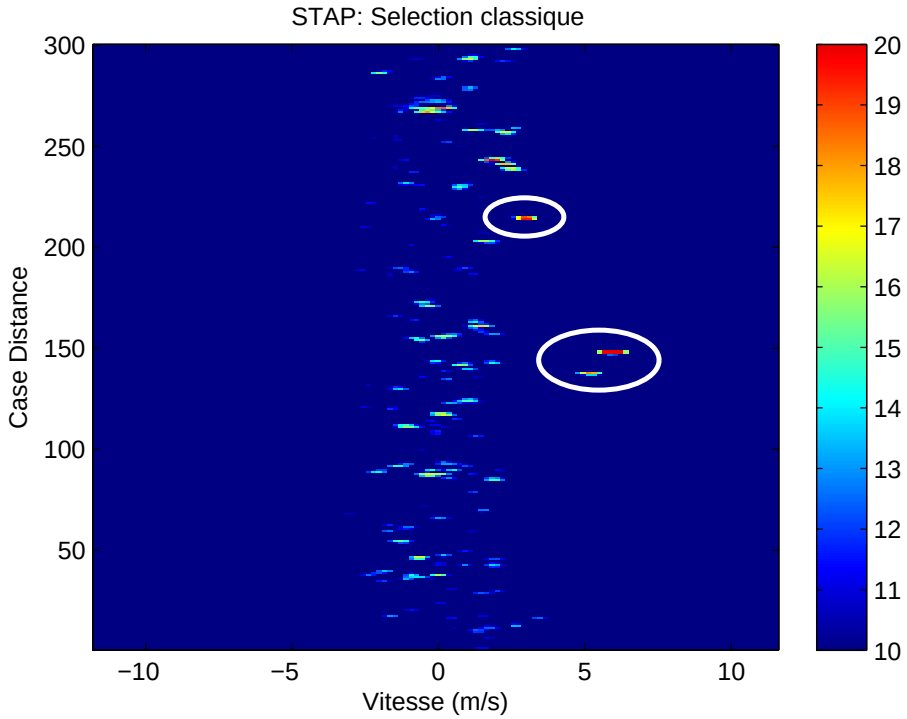


FIGURE 4.16 – Image vitesse-distance après STAP avec sélection classique

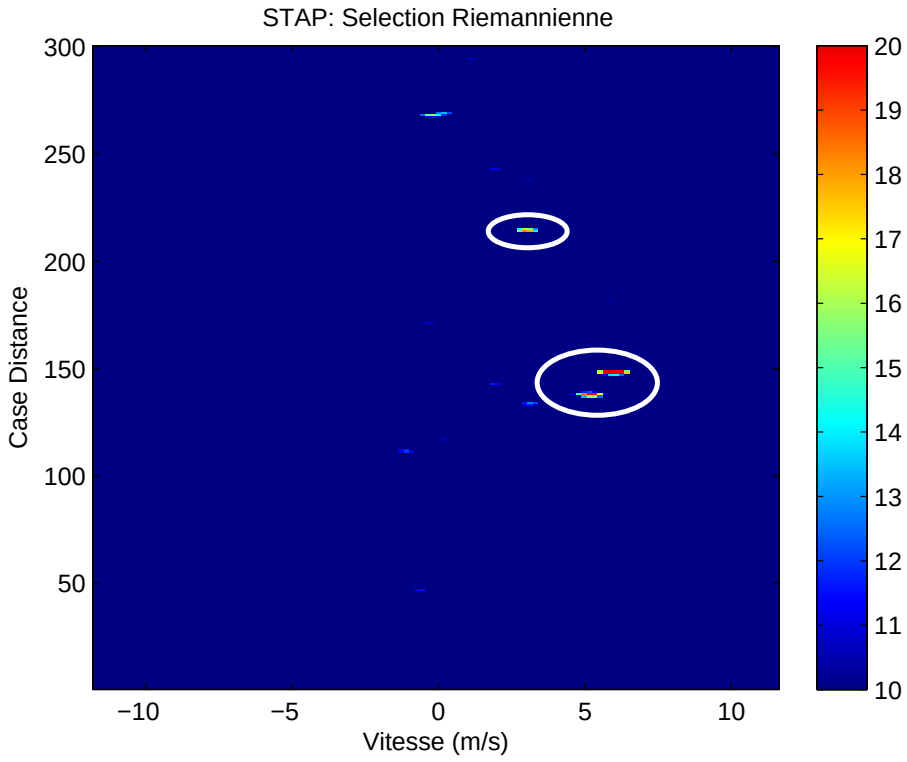


FIGURE 4.17 – Image vitesse-distance après STAP avec sélection Riemannienne

Pour la sélection des données d'entrainement, nous choisissons de nous restreindre à 20 cases distances qui entourent la case sous test tout en gardant 2 cases de garde autour de la case sous test. L'algorithme sélectionne les 10 cases distances dont les matrices de covariance sont les plus proches de la matrice de covariance de la case sous test. La contribution de la cible dans la matrice de covariance est négligeable.

Dans le cas de la distance Riemannienne qui est normalisée, nous fixons en plus un seuil à trois fois la distance moyenne entre deux cases distance de même taille et estimées sur le même nombre de points de manière à ne pas sélectionner les matrices qui dépassent ce seuil. Un minimum de deux matrices est sélectionné dans le cas où toutes les distances dépasseraient ce seuil.

Sur la figure 4.16 qui montre le traitement STAP avec sélection classique, nous voyons clairement les cibles mais aussi beaucoup de fausses alarmes dues à une mauvaise réjection du fouillis par le filtre STAP. Sur la figure 4.17, nous pouvons voir qu'avec la sélection en géométrie Riemannienne, il y a beaucoup moins de fausses alarmes alors que le niveau des cibles est très proche de la sélection classique : perte de 1dB sur la cible de la case distance 214 et gain de 0,5dB sur la cible de la case distance 138.

4.5 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche pour la sélection de données d'entrainement. La géométrie de l'information nous indique que la métrique Riemannienne est la bonne métrique lorsque nous travaillons avec des matrices Hermitiennes définies positives. En traitement du signal, son utilisation apporte des gains importants pour la sélection et le traitement des données d'entrainement. De plus, nous avons montré que d'un point de vue physique le critère optimal pour la sélection de données d'entrainement est une approximation de la distance en géométrie Riemannienne. Ce critère, qui est tout aussi performant, a l'avantage d'avoir un coût calculatoire réduit.

Nous avons montré que l'utilisation de la distance Riemannienne entre matrices de covariance pour sélectionner celles qui sont les plus proches des données primaires est efficace et adaptée à la sélection des données d'entrainement en traitement STAP. L'étude de la moyenne de matrices en géométrie Riemannienne que nous avons menée nous amène à espérer une amélioration supplémentaire des traitements STAP par rapport à l'utilisation de la moyenne arithmétique.

Troisième partie

Algorithmie et implémentation sur GPU

1

La charge calcultatoire des traitements STAP

Sommaire

L'augmentation de la complexité des systèmes radar ainsi que le traitement du signal associé impliquent un hausse significative de la charge calculatoire. Dans ce chapitre, nous analysons la complexité calculatoire des différents traitements STAP étudiés et développés dans la Partie 2.

1.1 Charge de calcul des traitements STAP

Dans les Sections suivantes, nous gardons les notations précédemment utilisées. Nous considérons pour les traitements STAP une antenne à N voies, M_p impulsions, M retard pour le filtre spatio-temporel ce qui implique $K_t = M_p - M + 1$ nombre d'échantillons pour l'estimation de la matrice de covariance par case distance, nb_{est} le nombre de cases distances utilisées pour l'estimation de la matrice et enfin p la dimension du sous-espace interférence pour les méthodes à rang réduit. Les opérations sont effectuées sur des nombres complexes [70] mais les résultats sont en opérations flottantes réelles (Flop) et en opérations flottantes réelles par secondes (Flop/s ou FLOPS).

1.1.1 Traitement STAP classique (FIR)

Estimation de la matrice de covariance

- estimation de $\mathbf{R}$ pour chaque case distance

$$\frac{\mathbf{X}\mathbf{X}^H}{K_t} \longrightarrow 8K_t(MN)^2 + 2(MN)^2$$

- moyenne arithmétique de $\mathbf{R}$ en distance

$$\frac{\mathbf{R_1} + \mathbf{R_2} + ...\mathbf{R_{nb_{est}}}}{nb_{est}} \longrightarrow 2(MN)^2(nb_{est} + 1)$$

- *diagonal loading*

$$\mathbf{R_{DL}} = \mathbf{R} + \delta\mathbf{I} \longrightarrow 2(MN)^2$$

Charge totale pour l'estimation de la matrice : $2(4K_t + nb_{est} + 3)(MN)^2$

Inversion de la matrice de covariance

- factorisation LU de la matrice :

$$\frac{8}{3}(MN)^3$$

- inversion de la matrice LU :

$$\frac{16}{3}(MN)^3$$

Charge totale pour l'inversion de la matrice : $8(MN)^3$

Calcul des poids du filtre $\mathbf{w}^H$

- nous négligeons le calcul du terme $\mathbf{w}^H\mathbf{R_{th}}$ qui est commun à toutes les cases distances
- calcul de $\mathbf{w}^H\mathbf{R_{th}}\mathbf{R}^{-1} \longrightarrow 8(MN)^2$
- calcul de $\mathbf{w}^H\mathbf{R_{th}}\mathbf{R}^{-1}\mathbf{R_{th}}\mathbf{w}^H \longrightarrow 8(MN)$

Charge totale pour le calcul des poids du filtre $8(MN)^2 + 8(MN)$

Application du filtre et transformée de Fourier

- Calcul de l'APR : $8(MN)^2 + 8(MN)$
- Calcul de $\mathbf{w}^H\mathbf{X}$: $8(MN)K_t$
- Calcul de la FFT Doppler : $5K_t \log_2(K_t)$

Charge totale pour l'application des poids et FFT : $8(MN)^2+8(K_t+1)(MN)+5K_t \log_2(K_t)$

Total

Pour une case distance, le nombre d'opérations à traiter est :

$$8(MN)^3 + 2(4K_t + nb_{est} + 11)(MN)^2 + 8(MN)(K_t + 2) + 5K_t \log_2(K_t)$$

1.1.2 Traitement *Eigencanceller* (EC)

Estimation de la matrice de covariance

- estimation de $\mathbf{R}$ pour chaque case distance

$$\frac{\mathbf{X}\mathbf{X}^H}{K_t} \longrightarrow 8K_t(MN)^2 + 2(MN)^2$$

- moyenne arithmétique de $\mathbf{R}$ en distance

$$\frac{\mathbf{R_1} + \mathbf{R_2} + ...\mathbf{R_{nb_{est}}}}{nb_{est}} \longrightarrow 2(MN)^2(nb_{est} + 1)$$

Charge totale pour l'estimation de la matrice : $2(4K_t + nb_{est} + 1)(MN)^2$

Décomposition en éléments propres de la matrice de covariance

- EVD :

$$23(MN)^3$$

Charge totale pour la décomposition en éléments propres : $23(MN)^3$

Calcul des poids du filtre $\mathbf{w}^H$

- calcul de $\mathbf{V_c}\mathbf{V_c^H} \longrightarrow 8p(MN)^2$
- calcul de $(\mathbf{I} - \mathbf{V_c}\mathbf{V_C^H}) \longrightarrow 2(MN)^2$
- calcul de $\mathbf{w^H} = \mathbf{s^H}(\mathbf{I} - \mathbf{V_c}\mathbf{V_c^H}) \longrightarrow 8(MN)^2$

Charge totale pour le calcul du filtre $\mathbf{w^H}$: $(10 + 8p)(MN)^2$

Application du filtre et transformée de Fourier

- Calcul de l'APR : $8(MN)^2 + 8(MN)$
- Calcul de $\mathbf{w}^H\mathbf{X}$: $8(MN)$
- Division $4(MN)$
- Calcul de la FFT Doppler : $5K_t \log_2(K_t)$

Charge totale pour l'application des poids et FFT : $8(MN)^2 + 20(MN) + 5K_t \log_2(K_t)$

Total

Pour une case distance, le nombre d'opérations à traiter est :

$$23(MN)^3 + 2(4K_t + nb_{est} + (18 + 8p))(MN)^2 + 20(MN) + 5K_t \log_2(K_t)$$

1.1.3 Traitement FAPI

Estimation

Pas d'estimation

Calcul de la base de vecteurs propres avec FAPI

– coût FAPI : $K_t(21(MN)p + 61p + 24p^2 + 32MN)$

Charge totale pour FAPI : $K_t(21(MN)p + 61p + 24p^2 + 32MN)$

Calcul des poids du filtre $\mathbf{w}^H$

– calcul de $\mathbf{W_c W_c^H} \longrightarrow 8p(MN)^2$
– calcul de $(\mathbf{I} - \mathbf{W_c W_C^H}) \longrightarrow 2(MN)^2$
– calcul de $\mathbf{w^H} = \mathbf{s^H}(\mathbf{I} - \mathbf{W_c W_c^H}) \longrightarrow 8(MN)^2$

Charge totale pour le calcul du filtre $\mathbf{w^H}$: $(10 + 8p)(MN)^2$

Application du filtre et transformée de Fourier

– Calcul de l'APR : $8(MN)^2 + 8(MN)$
– Calcul de $\mathbf{w}^H\mathbf{X}$: $8(MN)$
– Division $4(MN)$
– Calcul de la FFT Doppler : $5K_t \log_2(K_t)$

Charge totale pour l'application des poids et FFT : $8(MN)^2 + 20K_t(MN) + 5K_t \log_2(K_t)$

Total

Pour une case distance, le nombre d'opérations à traiter est :

$$(18 + 8p)(MN)^2 + [(52 + 21p)K_t](MN) + 24K_t p^2 + 61K_t p + 5K_t \log_2(K_t)$$

1.1.4 Traitement Stop-Band APES

Estimation de la matrice de covariance

– estimation de $\mathbf{R}$ pour chaque case distance

$$\frac{\mathbf{X}\mathbf{X}^H}{K_t} \longrightarrow 8K_t(MN)^2 + 2(MN)^2$$

– *diagonal loading*

$$\mathbf{R_{DL}} = \mathbf{R} + \delta\mathbf{I} \longrightarrow 2(MN)^2$$

Charge totale pour l'estimation de la matrice : $(8K_t + 4)(MN)^2$

Stop-Band et inversion (sans lemme d'inversion)

- cout de $\mathbf{G}(\mathbf{G^H G})^{-1}\mathbf{G^H} \longrightarrow 24(MN)^2 + 112(MN)$
- différence $\mathbf{Q} = \mathbf{R} - \mathbf{G}(\mathbf{G^H G})^{-1}\mathbf{G^H} \longrightarrow 2(MN)^2$
- inversion de $\mathbf{Q} \longrightarrow 8(MN)^3$

Charge totale pour l'inversion de la matrice : $K_t[8(MN)^3 + 26(MN)^2 + 112(MN)]$

Stop-Band et inversion (avec lemme d'inversion)

- inversion de $\mathbf{R} \longrightarrow 8(MN)^3$
- Cout de

$$\mathbf{R}^{-1}\mathbf{G}(\mathbf{G^H G} - \mathbf{G^H R}^{-1}\mathbf{G})^{-1}\mathbf{G^H R}^{-1} \longrightarrow 96(MN)^2 + 144(MN)$$

Charge totale pour l'inversion de la matrice : $8(MN)^3 + Kt[96(MN)^2 + 144(MN)$

Calcul des poids du filtre $\mathbf{w}^H$

- calcul de $\mathbf{w^H} = \mathbf{s^H Q} \longrightarrow 8(MN)^2$

Charge totale pour le calcul du filtre $\mathbf{w^H}$: $8K_t(MN)^2$

Application du filtre et transformée de Fourier

- Calcul de l'APR : $8(MN)^2 + 8(MN)$
- Calcul de $\mathbf{w}^H\mathbf{X}$: $8(MN)$
- Division $4(MN)$
- Calcul de la FFT Doppler : $5K_t \log_2(K_t)$

Charge totale pour l'application des poids et FFT : $8K_t(MN)^2+20K_t(MN)+5K_t \log_2(K_t)$

Total

Pour une case distance, le nombre d'opérations à traiter est :

- sans lemme : $8Kt(MN)^3 + (50K_t + 4)(MN)^2 + 132K_t(MN) + 5K_t \log_2(K_t)$
- avec lemme : $8(MN)^3 + (120Kt + 4)(MN)^2 + 264Kt(MN) + 5K_t \log_2(K_t)$

1.1.5 Traitement EC-Stop-Band

Estimation de la matrice de covariance

- estimation de $\mathbf{R}$ pour chaque case distance

$$\frac{\mathbf{X}\mathbf{X}^H}{K_t} \longrightarrow 8K_t(MN)^2 + 2(MN)^2$$

- *diagonal loading*

$$\mathbf{R_{DL}} = \mathbf{R} + \delta\mathbf{I} \longrightarrow 2(MN)^2$$

Charge totale pour l'estimation de la matrice : $(8K_t + 4)(MN)^2$

Décomposition en éléments propres de la matrice de covariance

- cout du $\mathbf{G}(\mathbf{G^H G})^{-1}\mathbf{G^H} \longrightarrow 24(MN)^2 + 112(MN)$
- différence $\mathbf{Q} = \mathbf{R} - \mathbf{G}(\mathbf{G^H G})^{-1}\mathbf{G^H} \longrightarrow 2(MN)^2$
- coût de l'EVD : $23(MN)^3$

Charge totale pour l'EVD de la matrice : $K_t(23(MN)^3 + 26(MN)^2 + 112(MN))$

Calcul des poids du filtre $\mathbf{w}^H$

- calcul de $\mathbf{V_c V_c^H} \longrightarrow 8p(MN)^2$
- calcul de $(\mathbf{I} - \mathbf{V_c V_C^H}) \longrightarrow 2(MN)^2$
- calcul de $\mathbf{w^H} = \mathbf{s^H}(\mathbf{I} - \mathbf{V_c V_c^H}) \longrightarrow 8(MN)^2$

Charge totale pour le calcul du filtre $\mathbf{w^H}$: $K_t(10 + 8p)(MN)^2$

Application du filtre et transformée de Fourier

- Calcul de l'APR : $8(MN)^2 + 8(MN)$
- Calcul de $\mathbf{w}^H\mathbf{X}$: $8(MN)$
- Division $4(MN)$
- Calcul de la FFT Doppler : $5K_t \log_2(K_t)$

Charge totale pour l'application des poids et FFT : $8K_t(MN)^2 + 20K_t(MN) + 5K_t \log_2(K_t)$

Total

Pour une case distance, le nombre d'opérations à traiter est :

$$23K_t(MN)^3 + ((58 + 8p)K_t + 2)(MN)^2 + (132K_t)(MN)$$

1.1.6 Traitement FAPI-Stop-Band

Suppression du signal d'intérêt des données

- coût de $\mathbf{P} = \mathbf{S}(\mathbf{S^H S})^{-1}\mathbf{S^H} \longrightarrow 24(MN)^2 + 112(MN)$
- calcul de $(\mathbf{I} - \mathbf{P}) \longrightarrow 2K_t^2$
- calcul de $\mathbf{Y} = \mathbf{X}(\mathbf{I} - \mathbf{P}) \longrightarrow 8(MN)K_t^2$

Charge totale pour la suppression du signal : $24(MN)^2 + (112 + 8K_t^3)(MN) + 2K_t^3$

Calcul de la base de vecteurs propres avec FAPI

- coût FAPI : $K_t(21(MN)p + 61p + 24p^2 + 32MN)$

Charge totale pour FAPI : $K_t^2(21(MN)p + 61p + 24p^2 + 32MN)$

Calcul des poids du filtre $\mathbf{w}^H$

- calcul de $\mathbf{W_c W_c^H} \longrightarrow 8p(MN)^2$
- calcul de $(\mathbf{I} - \mathbf{W_c W_C^H}) \longrightarrow 2(MN)^2$
- calcul de $\mathbf{w^H} = \mathbf{s^H}(\mathbf{I} - \mathbf{W_c W_c^H}) \longrightarrow 8(MN)^2$

Charge totale pour le calcul du filtre $\mathbf{w^H}$: $K_t(10 + 8p)(MN)^2$

Application du filtre et transformée de Fourier

- Calcul de l'APR : $8(MN)^2 + 8(MN)$
- Calcul de $\mathbf{w}^H\mathbf{X}$: $8(MN)$
- Division $4(MN)$
- Calcul de la FFT Doppler : $5K_t \log_2(K_t)$

Charge totale pour l'application des poids et FFT : $8K_t(MN)^2+20K_t(MN)+5K_t \log_2(K_t)$

Total

Pour une case distance, le nombre d'opérations à traiter est :

$$(K_t(18+8p))(MN)^2 + [(40+21p)K_t^2 + 20K_t + 112](MN) + 6Kt^2 + 24p^2 + 61p$$

1.2 Résultats sur diverses configurations

Dans cette Section, nous comparons les charges de calcul des différents algorithmes pour une configuration de radar GMTI proche de celle utilisée dans les données **ONERA-AS**. L'antenne possède donc $N = 8$ voies de réception, nous prenons $M = 5$ retards pour former le vecteur STAP sur les $M_p = 32$ impulsions du radar. La fréquence de répétition des impulsions est $f_{PRF} = 4\,kHz$ et les données sont composées de 3750 cases distance.

Charge calculatoire STAP/GMTI (nombre opérations flottantes réelles/seconde)

- **Antenne 8 voies**
- **PRF 4KHz - Mp=32 impulsions - M=5 points**
- **K=3750 cases distance**

SMI (FIR)	**444 GFlops/s**	Estimation matrice	187 GFlops/s
		Inversion matrice	240 GFlops/s
		Calcul des poids	6 GFlops/s
		Filtrage + FFT	10 GFlops/s
EC-STOP-BAND	**22667 GFlops/s**	Estimation matrice	170 GFlops/s
		EVD	19925 GFlops/s
		Calcul des poids	2394 GFlops/s
		Filtrage + FFT	179 GFlops/s
FAPI-STOP-BAND	**12153 GFlops/s**	Retrait signal	3315 GFlops/s
		FAPI	6266 GFlops/s
		Calcul des poids	2394 GFlops/s
		Filtrage + FFT	179 GFlops/s
STOP-BAND	**7842 GFlops/s**	Estimation matrice	171 GFlops/s
		Inversion matrice	7325 GFlops/s
		Calcul des poids	168 GFlops/s
		Filtrage	179 GFlops/s
STOP-BAND (lemme)	**2030 GFlops/s**	Estimation matrice	171 GFlops/s
		Inversion matrice	1513 GFlops/s
		Calcul des poids	168 GFlops/s
		Filtrage + FFT	179 GFlops/s

FIGURE 1.1 – Tableau des charges de calcul pour les différents traitements : STAP FIR (référence), Stop-Band, Stop-Band avec lemme d'inversion matriciel, EC-Stop-Band et FAPI-Stop-Band

Charge calculatoire STAP/GMTI (nombre opérations flottantes réelles/seconde)

- **Antenne 8 voies**
- **PRF 4KHz - Mp=32 impulsions - M=5 points**
- **K=3750 cases distance**

EC	**895 GFlops/s**	Estimation matrice	187 GFlops/s
		EVD	690 GFlops/s
		Calcul des poids	7,5 GFlops/s
		Filtrage + FFT	10 GFlops/s
FAPI	**508 GFlops/s**	Estimation	0 GFlops/s
		FAPI	497 GFlops/s
		Calcul des poids	7,5 GFlops/s
		Filtrage + FFT	10 GFlops/s

FIGURE 1.2 – Tableau des puissance de calcul pour les différentes traitement : STAP FIR (référence), *Eigencanceller* et FAPI

Les figure 1.1 et 1.2 indiquent les puissances de calcul pour les différentes méthodes étudiées dans les chapitres précédents. Les opérations sont effectuées sur des nombres complexes mais les puissances de calcul sont exprimées en nombre d'opérations flottantes réelles [70] par seconde. Cette puissance est calculée de la façon suivante :

$$\texttt{Puissance(GFlop/s)} = \texttt{Charge(Flop)}/(\texttt{T} * 10^9)$$

Nous voyons que les puissances de calcul des méthodes non basées sur Stop-Band sont de l'ordre de quelques centaines de GFlop/s. Les méthodes à données primaires seules, basées sur Stop-Band, ont une charge plus élevée puisqu'il faut calculer un filtre pour chaque case Doppler. Leur coût allant de 2 TFlop/s à plus de 22 TFlop/s. L'apport de l'algorithme FAPI varie d'un facteur 1,5 à un facteur 2. Ici, nous avons utilisé le rang de Brennan (voir Partie 2, Chap. 2.2.4) qui est très généralement surestimé comme référence pour le rang p et la charge de calcul de l'algorithme FAPI dépend fortement de ce paramètre.

La Figure 1.3 montrent les puissances de calcul pour une configuration Air-air des méthodes STAP classique, Stop-Band, Stop-Band avec lemme d'inversion matriciel, EC et FAPI. Les résultats sont du même ordre que pour la configuration précédente, c'est à dire quelques centaines de GFlop/s pour un traitement standard et des puissances nécéssaires supérieures à 1 TFlop/s pour les méthodes Stop-Band et pour les méthodes à rang réduit.

Charge calculatoire STAP/Air-Air (nombre opérations flottantes réelles/seconde) ➢ Antenne 16 voies ➢ MFR 20KHz - Mp=128 impulsions - M=8 points ➢ K=125 cases distance			
SMI (FIR)	**647,58 GFlop/s**	Estimation matrice	312,32 GFlop/s
		Inversion matrice	327,68 GFlop/s
		Calcul des poids	2,58 GFlop/s
		Filtrage	5,00 GFlop/s
EC	**3064 GFlop/s**	Estimation matrice	311,68 GFlop/s
		EVD	942,0 GFlop/s
		Calcul des poids	1808 GFlop/s
		Filtrage	2,43 GFlop/s
FAPI	**3581 GFlop/s**	Estimation matrice	0 GFlop/s
		FAPI	1770 GFlop/s
		Calcul des poids	1808 GFlop/s
		Filtrage	2,43 GFlop/s
FAPI (p=MN/2)	**1791 GFlop/s**	Estimation matrice	0 GFlop/s
		FAPI	885.26 GFlop/s
		Calcul des poids	904 GFlop/s
		Filtrage	2,43 GFlop/s
STOP-BAND	**41626 GFlop/s**	Estimation matrice	311,04 GFlop/s
		Inversion matrice	40 690 GFlop/s
		Calcul des poids	309,76 GFlop/s
		Filtrage	315,81 GFlop/s
STOP-BAND (lemme)	**3594 GFlop/s**	Estimation matrice	311,04 GFlop/s
		Inversion matrice	2658 GFlop/s
		Calcul des poids	309,76 GFlop/s
		Filtrage	315,81 GFlop/s
FFT Doppler	0,082 GFlop/s		

FIGURE 1.3 – Tableau des charges de calcul pour les différentes traitement : STAP FIR (référence), Stop-Band, Stop-Band avec lemme d'inversion matriciel, EC et FAPI

1.3 Conclusion

Les puissances de calcul nécessaires à l'application d'un traitement STAP, même classique, dépassent largement les puissances de calcul des micro-processeurs généralistes qui atteignent une centaine de GFlop/s pour les modèles les plus performants. En revanche, la famille des processeurs dédiés au traitement graphique, appelées GPU pour *Graphics Processor Unit*, possèdent des puissances de calcul théoriques 10 à 20 fois supérieures aux processeurs généralistes rendant théoriquement possible l'utilisation de traitements STAP temps-réel, du moins pour un traitement classique. Dans le chapitre suivant, nous étudierons l'architecture de ces nouveaux processeurs afin de déterminer des méthodes de programmation efficaces pour nos traitements STAP.

2

Les processeurs GPU

Sommaire

Les processeurs généralistes (CPU) ont traditionnellement été au coeur des ordinateurs et des systèmes dédiés au calcul scientifique, mais nous avons observé ces dernières années une forte augmentation de l'utilisation de processeurs accélérateurs, comme les processeurs graphiques (GPU). Ces processeurs, initialement conçus pour calculer en temps réel des images en trois dimensions, sont utilisés parce qu'ils ont une consommation réduite et qu'ils offrent plus de puissance de calcul que des processeurs CPU équivalents. Ils sont aussi bien moins couteux, puisque plusieurs CPUs sont nécessaires pour égaler la performance d'un GPU.
Bien que ces processeurs soient très puissants pour des calculs sur des nombres flottants, leur capacités sont limitées du fait de leur architecture moins sophistiquée. Ils ne peuvent par exemple, ouvrir un fichier ou lancer un système d'exploitation sans l'aide d'un processeur CPU. Ainsi, la plupart des applications peuvent les utiliser en tant qu'accélérateur pour effecteur les calcul lourds, tandis que l'exécution des tâches complexes reste sur le CPU. Comme nous l'avons vu dans le chapitre précédent, les traitements STAP requièrent des puissances de calcul très importantes. Dans ce chapitre, nous étudions l'architecture d'un GPU moderne et son intérêt pour les traitements STAP.

2.1 Historique

Les microprocesseurs généralistes fondés sur une unique unité centrale de traitement ou *central processing unit*, comme les familles de processeurs Intel ou AMD ont vu leurs performances significativement augmenter et leur coût baisser au cours des 20 dernières années. Ces processeurs ont amené des puissances de calcul de milliards d'opérations (GFlop/s) sur des nombres flottants dans les ordinateurs personnels, et des centaines de GFlop/s dans les serveurs de calcul alors que la demande en puissance de calcul, notamment pour les applications multimédia, est toujours croissante. Durant cette période, les développeurs se sont appuyés sur les avancées matérielles pour augmenter la vitesse de leurs applications. Ainsi, le même programme s'exécute plus rapidement à chaque nouvelle génération de processeurs. Cependant, cette tendance s'est ralentie depuis 2003, à cause de la consommation énergétique et des problèmes de dissipation thermique qui ont limité l'augmentation de la fréquence d'horloge des processeurs ainsi que le nombre d'instructions qui peut être effectué à chaque période d'horloge dans un CPU. Tous les fabricants de microprocesseurs ont alors proposé des modèles intégrants plusieurs unités de traitement, appelés cœurs de traitement, qui sont utilisés sur chaque puce pour augmenter la puissance de traitement. Cette évolution a eu un impact important sur la communauté des développeurs [71]. Auparavant, la majorité des applications étaient écrites pour s'exécuter séquentiellement. L'exécution de ces programmes peut être comprise facilement en lisant les lignes de code une à une à la suite. A l'arrivée d'une nouvelle génération de processeurs, ces programmes tournaient simplement plus vite en exécutant plus rapidement les instructions contenues dans le code. Aujourd'hui, cela n'est plus tout à fait vrai, puisqu'un programme séquentiel ne s'exécutera que sur un seul cœur de processeur. Au contraire, les programmes parallèles, c'est à dire dans lesquels plusieurs tâches s'exécutent au même moment pour achever le travail plus rapidement, utiliseront plusieurs cœurs et s'exécuteront plus rapidement. Cette programmation n'est pas nouvelle puisque ce type de programmation est utilisé depuis des décennies par les serveurs de calcul scientifiques qui étaient composés de centaines ou de milliers de processeurs.

Depuis 2003, nous assistons à deux grandes tendances dans les architectures de processeurs. Les processeurs CPU multi-cœurs cherchent à maintenir la vitesse d'exécution des programmes séquentiels tout en augmentant le nombre de cœurs de traitement. Aujourd'hui un processeur CPU haut de gamme possède 4 à 6 cœurs, chaque cœur étant de type *out-of-order* à instructions multiples, supporte les instructions x86 (voir Annexe) et est conçu pour maximiser la vitesse d'exécution des programmes séquentiels. Au contraire, les architectures dites *many-cœurs*, comme les processeurs graphiques GPU, se concentrent sur l'augmentation des débits d'exécution de tâches largement parallèles. Les architectures *many-coeurs* sont construites autour d'un grand nombre de petits cœurs de calcul. Par exemple, un GPU haut de gamme possède aujourd'hui plus de 1000 cœurs, chacun de ces coeurs étant de type *in-order*, à instruction simple et partagent leur unité de contrôle et leur mémoire cache avec d'autres cœurs. Les processeurs *many-cœurs*, et plus particulièrement les GPU, sont les processeurs informatiques les plus puissants en calcul sur nombres flottants. Alors que l'augmentation des performances des processeurs généralistes CPU s'est ralentie, les GPUs ont vu leur performances fortement augmenter (voir Figure 2.1).

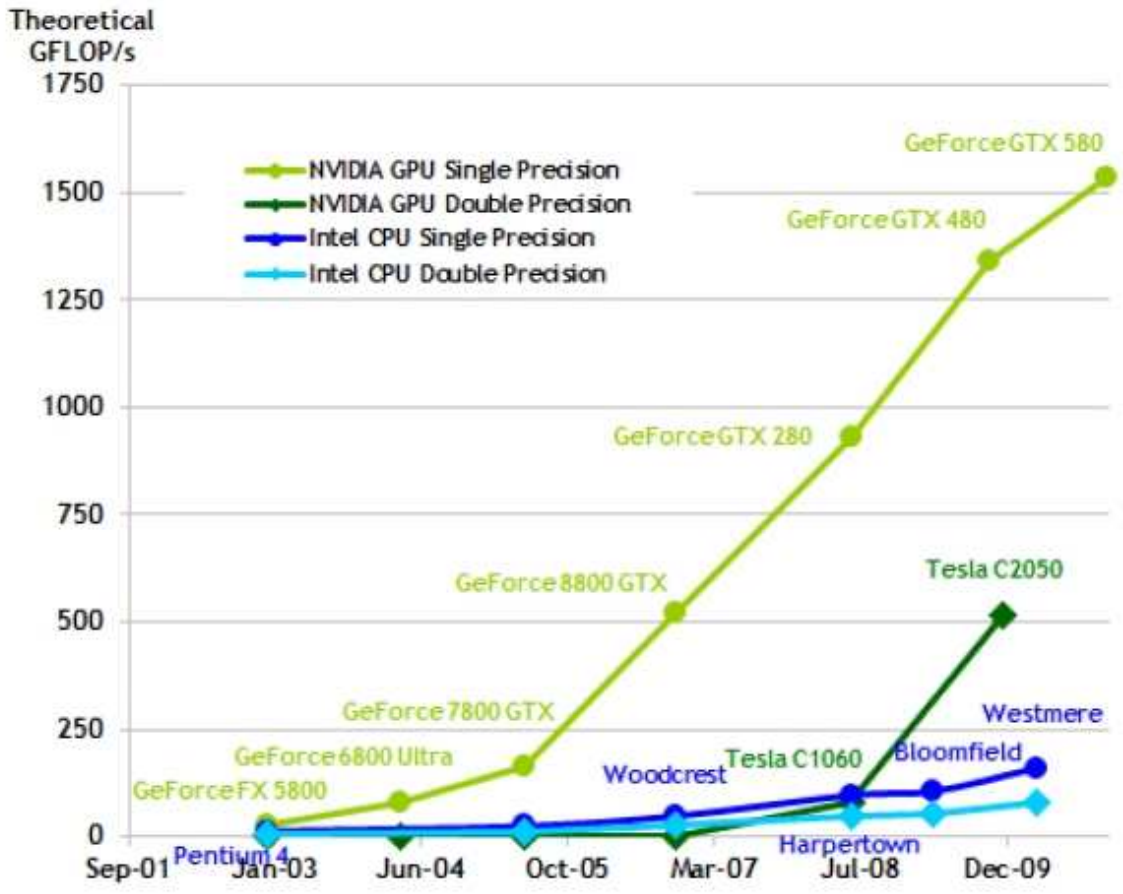


FIGURE 2.1 – Puissance de calcul théorique des GPU du fabriquant NVIDIA (vert) et des CPU du constructeur Intel (bleu) au cours du temps. Source : NVIDIA.

Aujourd'hui le rapport de puissance de calcul entre CPU et GPU est d'environ 1 pour 10 [72]. Nous pouvons nous demander pourquoi il existe une telle différence de puissance de calcul entre les GPUs à architecture *many-coeurs* et les processeurs généralistes multi-coeurs CPU. La réponse vient du design fondamentalement différent de ces deux types de microprocesseurs.

L'architecture d'un CPU est optimisée pour les performances de codes séquentiels. Pour cela, il utilise une unité de contrôle logique très sophistiquée qui permet aux instructions d'une tâche de s'exécuter sur plusieurs unité arithmétiques, ou même dans un ordre différent de leur séquence, tout en maintenant l'apparence d'une exécution séquentielle. Une mémoire cache importante est requise pour réduire la latence d'accès à la mémoire et aux instructions des applications complexes. Ni ces unités de contrôle logique ni la mémoire cache ne contribuent à l'augmentation de la puissance de calcul brute.

L'architecture des GPUs est quant à elle façonnée par l'industrie grandissante des jeux vidéos, qui depuis l'apparition des graphismes en 3D au milieu des années 1990, lui impose de pouvoir effectuer une quantité massive de calcul à virgule flottante pour chaque image dans les jeux les plus avancés.

Cette demande a conduit les concepteurs de GPUs à maximiser la surface de la puce ainsi que sa consommation dédiée aux calculs à virgule flottante. La solution consiste à optimiser le processeur pour l'exécution d'un nombre maximum de tâches. Les unités de calcul profitent du nombre très élevé de tâches pour travailler alors que certaines tâches attendent pour des accès mémoire, minimisant ainsi le recours à une grande quantité de mémoire cache ainsi qu'à une unité de contrôle logique sophistiquée. Une mémoire cache en petite quantité est tout de même présente pour éviter des aller-retours vers la mémoire centrale (DRAM). Finalement, une partie beaucoup plus importante de la puce est dédiée aux calculs à virgule flottante comme le montre la Figure 2.2.

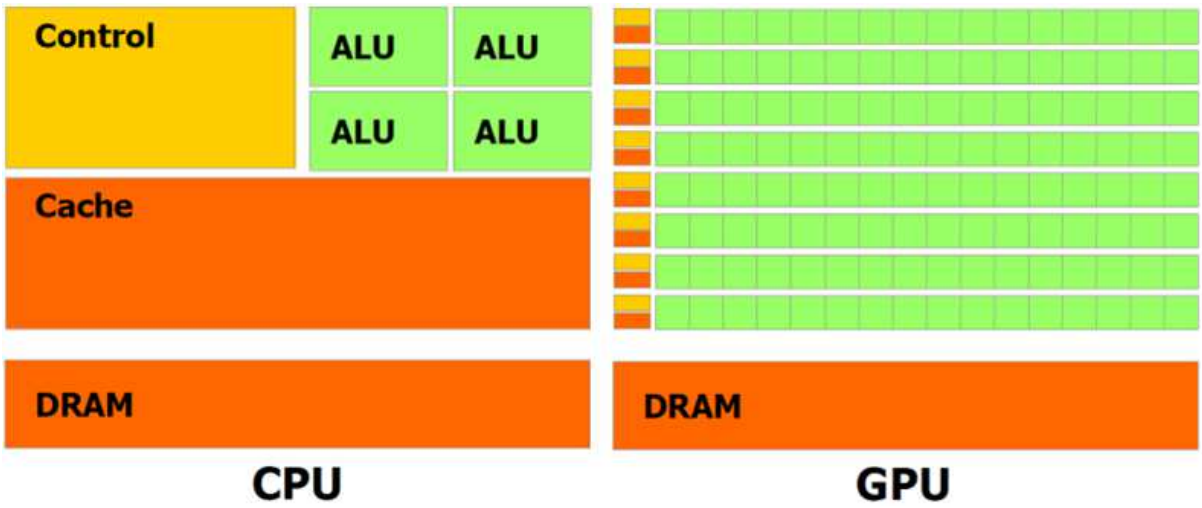


FIGURE 2.2 – Architecture d'un CPU et d'un GPU. En orange les mémoires, en jaune les unités de contrôle et en vert les unités de calcul ALU. Source : NVIDIA.

Pour simplifier, on attend d'un CPU qu'il exécute une tâche le plus rapidement possible alors que l'on attend d'un GPU de pouvoir traiter un maximum de tâches, ou plutôt une tâche sur un maximum de données, dans un temps réduit. Bien entendu un GPU doit également aller vite et un CPU doit traiter plusieurs tâches, mais l'évolution de leurs architectures, tout du moins jusqu'à aujourd'hui, a reflété cette priorité. Cela signifie dans le cas d'un GPU une multiplication des unités de traitement et dans le cas du CPU une complexification des unités de contrôle ainsi qu'une augmentation régulière de la mémoire cache embarquée.

Il est donc clair que les GPUs sont conçus comme des processeurs de calcul numérique, et qu'ils fonctionneront mal sur certaines tâches sur lesquelles les CPUs sont conçus pour être efficaces et les applications doivent être conçues pour effectuer les parties séquentielles sur CPU et les parties parallèles sur GPU. Les performances ne sont pas le seul facteur à prendre en compte pour une application scientifique. En particulier, le processeur choisi doit être présent en quantité sur le marché, afin de maintenir des coûts de production réduits et une vaste communauté de développeurs. En raison de leur présence dans tous les ordinateurs PC, des centaines de millions de GPUs sont actuellement en circulation dans le monde.

Jusqu'en 2006, les processeurs graphiques étaient difficiles à utiliser pour le calcul scientifique parce que les programmeurs devaient utiliser des interfaces de programmation (API) graphiques, comme OpenGL ou Direct3D, pour accéder à la puce. Cette technique est appelée GPGPU pour *General Purpose programming using a Graphics Processor Unit*. Les possibilités de calcul scientifique étaient grandement limitées par les APIs graphiques et seuls quelques programmeurs parvenaient à correctement les exploiter pour tirer profit de la puissance des GPUs. Cela amena la société NVIDIA, l'un des deux fabricants de GPUs, à proposer en 2007 un environnement et un langage de programmation appelé CUDA spécialement conçu pour pouvoir exécuter des calculs scientifiques sur GPU en s'affranchissant des interfaces de programmation graphique.

2.2 Architecture moderne d'un GPU

Nous décrivons ici l'architecture d'un GPU moderne, plus spécifiquement du GPU sur lequel ont été testées les performances d'un algorithme STAP. Introduit en 2010, l'architecture Fermi de NVIDIA est la seconde génération de GPU supportant le langage de programmation CUDA et la première génération de GPU à avoir pris en compte l'aspect calcul scientifique sur GPU lors de sa conception [73].

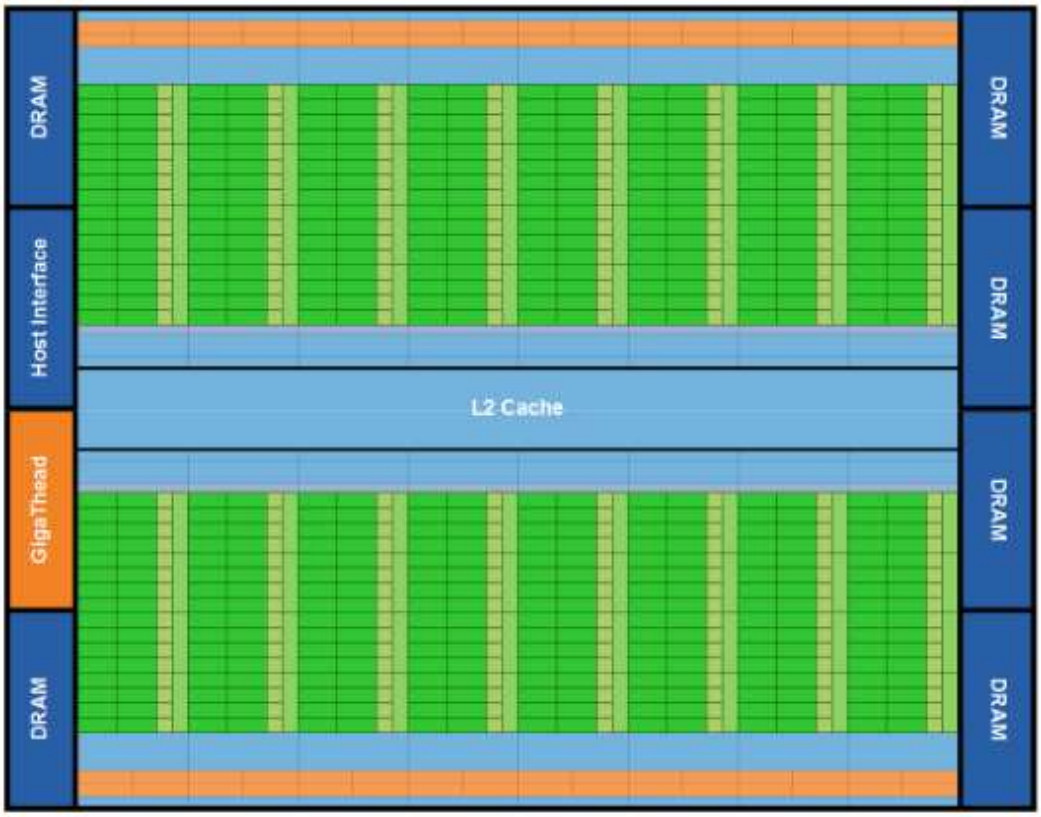


FIGURE 2.3 – Schéma d'un GPU "GF100" complet. Chaque multiprocesseur est un des rectangles verticaux contenant des unités de calcul (vert), une mémoire cache L1 et des registres (bleu clair) ainsi que des unités de contrôle (orange)

Le processeur "GF100", premier GPU à architecture Fermi, est composé de 512 coeurs de calcul, organisés en 16 groupes de 32 cœurs (voir Figure 2.3), ces groupes sont appelés multiprocesseurs ou *Stream Multiprocessor* (SM). Comme nous allons le voir, ces cœurs ne sont pas comparables aux coeurs d'un CPU.

2.2.1 Organisation de la mémoire

Chaque multiprocesseur de Fermi dispose d'une mémoire cache de premier niveau (L1) programmable qui va servir autant de cache L1 que de mémoire partagée. Ce cache de 64 Ko va fonctionner avec 48 Ko pour l'un et 16 Ko pour l'autre, au cas par cas. 64 Ko en mode L1 ou en mode mémoire partagée ou encore 32 Ko et 32 Ko ne sont pas permis. Il est donc possible soit de profiter de 48 Ko de L1, soit de se limiter à 16 Ko de L1 et de disposer de plus de groupes de tâches (chacun ayant besoin de sa propre mémoire partagée) dans le multiprocesseur pour mieux masquer les latences. Après ce cache L1 programmable, nous trouvons une mémoire cache de second niveau L2 cohérente et unifiée de 128 Ko par contrôleur mémoire, soit 768 Ko au total. Les opérations de stockage passent par ces 2 caches avant de se résoudre à accéder à la mémoire globale DRAM, extérieure au GPU et qui a donc une latence beaucoup plus importante.

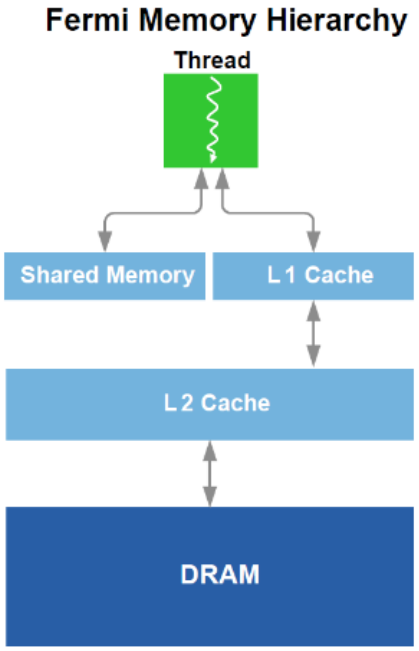


FIGURE 2.4 – Hiérarchie de la mémoire sur un GPU Fermi. Source : NVIDIA.

L'avantage est de pouvoir récupérer ces données plus rapidement si elles sont en cache mais aussi de pouvoir réduire l'incidence sur les performances d'un dépassement du nombre de registres disponibles sur le GPU en les stockant dans la mémoire cache. Auparavant les registres supplémentaires devaient résider en mémoire globale DRAM, avec une latence très longue, en contradiction avec le rôle d'un registre.

2.2.2 Les unités de calcul

Chacun des 16 multiprocesseurs dispose d'un double ordonnanceur et de 4 blocs d'exécutions qui fonctionnent à deux fois la fréquence du reste du GPU [74]. Ces blocs d'exécution sont composés de :

- deux unités SIMD 16-way (les 32 cœurs) capablent de traiter 32 opérations FMA FP32, 32 ADD INT32, 16 MUL INT32, 16 FMA FP64
- une unité SFU quadruple pouvant traiter 4 fonctions spéciales (cos, sqrt...) FP32 ou 16 interpolations
- une unité Load/Store 16-way 32 bits

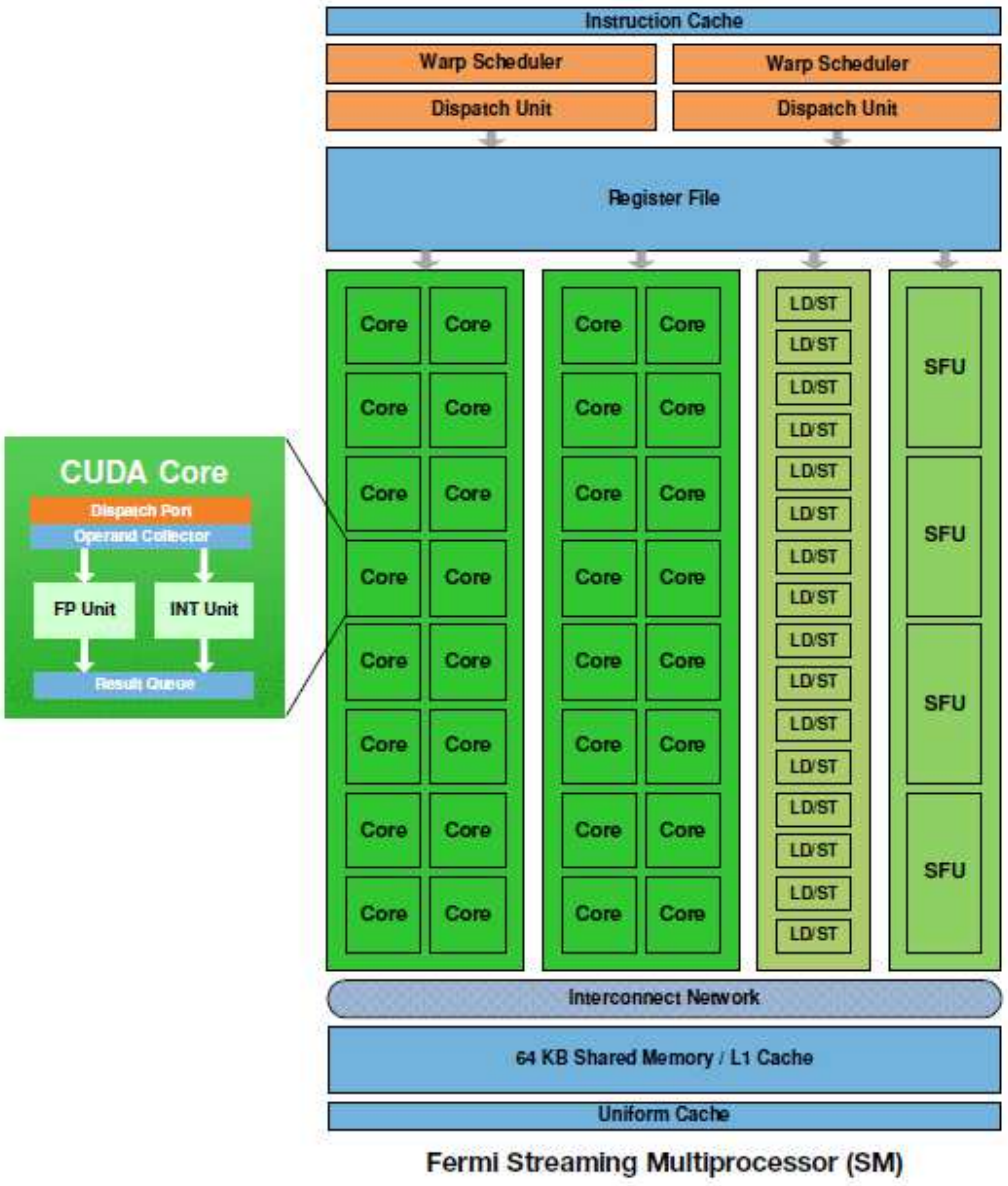


FIGURE 2.5 – Structure d'un multiprocesseur de l'architecture GPU Fermi. Source : NVIDIA.

Les unités de calcul du GPU peuvent traiter des instructions FMA (*fused multiply add*) contrairement aux processeurs généralistes CPU qui sont limités aux instructions MAD (*multiply add*). La différence est la précision des calculs, une instruction MAD se comporte réellement comme une multiplication flottante suivie d'une addition flottante, le résultat est donc arrondi à chaque étape. L'instruction FMA par contre conserve la précision intermédiaire et n'effectue l'arrondi qu'à la fin du calcul, et le GPU supporte donc complètement la norme IEEE 754-2008 [75].

Les deux unités SIMD 16-way sont distinctes et travaillent sur une instruction différente. La première unité peut ainsi exécuter 16 FMA FP32 alors que la seconde traite 16 ADD INT32. L'unité SFU, qui traite les fonctions spéciales, quadruple est découplée et l'ordonnanceur peut donner des instructions aux 2 unités SIMD une fois qu'elle est au travail, ce qui permet donc de faire travailler les SFUs en même temps que les SIMDs. A la vue de cette architecture, nous voyons que nous ne pouvons pas comparer les 512 cœurs du GPU aux cœurs d'un CPU. Un cœur CPU possède une unité de traitement vectorielle et peut traiter indépendamment n'importe quelles instructions. Il est plus proche d'un multiprocesseur GPU que d'une unité de traitement GPU. Un GPU Fermi peut donc être vu comme étant composé de 16 puissants cœurs de calcul.

2.2.3 Débit mémoire et communication avec le CPU

Comme nous l'avons vu dans la Section 2.2.1, le GPU possède plusieurs niveaux de mémoire. Les mémoires caches se trouvent sur le processeur lui-même alors que la mémoire globale DRAM, beaucoup plus importante de 3 à 6 Go contre quelques Ko pour les mémoires caches, se trouve sur des puces externes au GPU, de la même manière que la mémoire centrale DRAM d'un processeur généraliste CPU, aussi appelée mémoire vive. Toujours à cause d'approches différentes, le débit de la mémoire DRAM des GPU est beaucoup plus important que celui de la mémoire DRAM du CPU (20 à 30 Go/s sur CPU, 100 à 200 Go/s sur GPU).

En revanche, le GPU doit être piloté par le CPU et les données devant être traitées par le GPU doivent être transférées de la mémoire centrale du système vers la mémoire globale du GPU. Cette communication passe par le port PCI Express qui, dans sa version 2.0, autorise un débit maximum théorique de 8 Go/s. Ce débit étant relativement lent, il convient d'effectuer le moins de transfert possible et donc, d'exécuter le plus de calculs sur GPU avant de rapatrier les résultats sur CPU. Il existe cependant un moyen de contourner en partie ce problème en utilisant des transferts asynchrones, c'est à dire que les données sont transférées pendant que d'autres données sont traitées.

2.2.4 Puissance de calcul

Nous avons calculé les puissances de calcul maximales de différentes architectures avec quelques instructions courantes : multiplication+addition (MAD), multiplication+addition fusionnés (FMA), addition (ADD) et multiplication (MUL) sur 3 types de nombres : flottants simple précision (FP32), flottants double précision (FP64) et entiers (INT32).

Pour Fermi, nous considérons que les 16 SM sont actifs et fonctionnent à une fréquence de 1500 MHz, c'est à dire que le GPU fonctionne à 750 MHz. Nous le comparons au GPU concurrent Cypress [76] de la société AMD/ATI (voir Annexe C.1) ainsi qu'à la puissance maximale autorisée par les unités vectorielles (voir Annexe C.2) d'un processeur CPU Intel Core i7-975 [77] qui est très proche du CPU sur lequel nous implanterons le traitement.

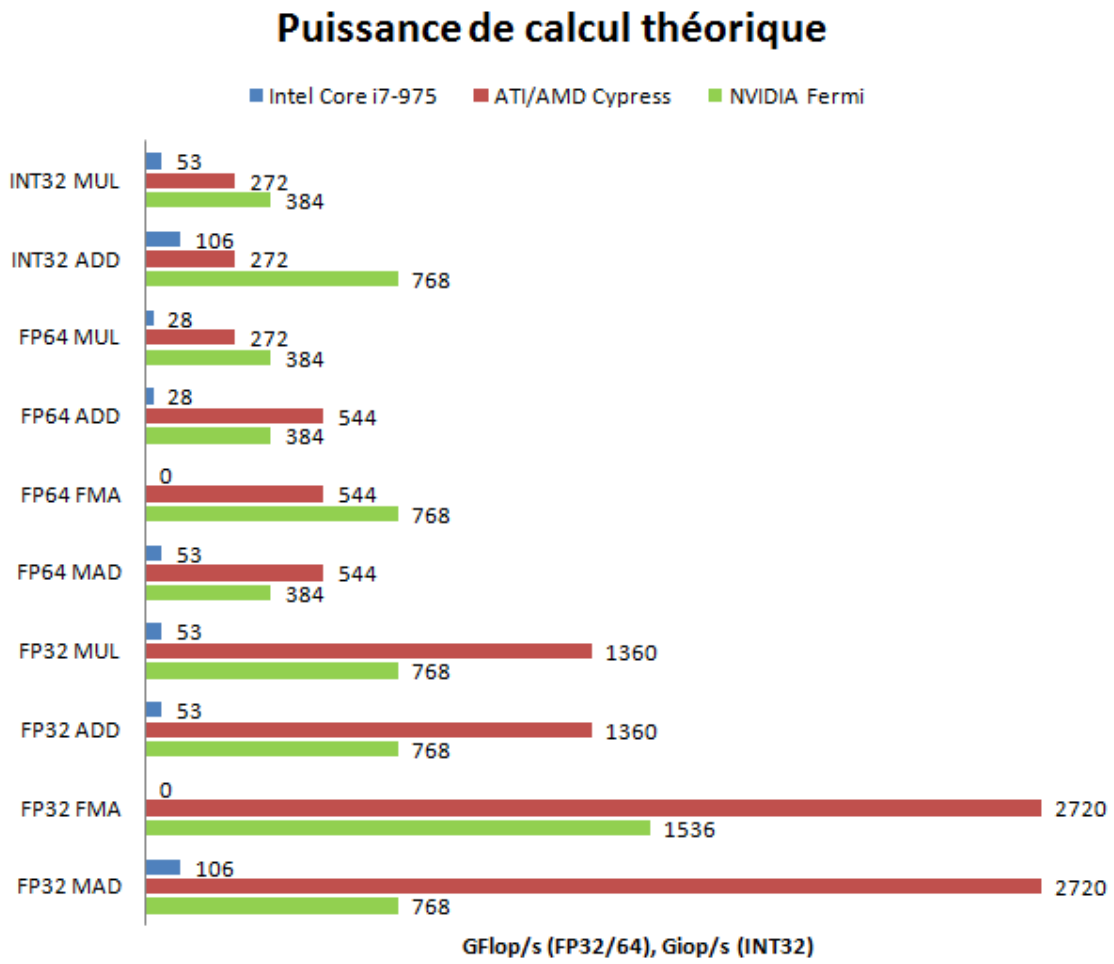


FIGURE 2.6 – Comparaison des puissances de calcul théoriques du GPU NVIDIA Fermi (vert), du GPU AMD/ATI Cypress (rouge) et du CPU Intel Corei7-975 (bleu).

La Figure 2.6 montre que les deux GPUs sont beaucoup plus rapides que le CPU sur toutes les instructions et notamment sur les multiplications+addition (MAD/FMA) qui sont à la base du calcul matriciel et vectoriel (produits scalaires, multiplications matrice-matrice, vecteur-matrice) où les GPUs sont environ 15 à 27 fois plus rapides sur les nombres flottants en simple précision et 10 à 15 fois plus rapides en double précision. Par rapport à Fermi, Cypress affiche une puissance de calcul supérieure en simple précision flottante mais en pratique il est plus difficile de l'atteindre compte tenu de son architecture et de l'interface de programmation que propose le constructeur (voir Annexe C.1). En double précision, le GPU Fermi reste cependant plus puissant.

2.3 Programmation : le langage CUDA

Introduit en 2007 par NVIDIA, CUDA est vu comme un ensemble d'extensions au C. En effet, cette technologie est fortement basée sur le langage C. Ceci permet aux programmeurs déjà initiés de s'adapter rapidement à la programmation parallèle. Par rapport à son alternative libre OpenCL [78], CUDA est beaucoup plus développé, plus simple et contient nativement la plupart des librairies de calcul. Nous décrivons ici très brièvement ces extensions.

2.3.1 Les fonctions

Les programmes CUDA utilisent conjointement des fonctions et des variables en C et en CUDA. Des déclarations permettent de les dissocier :

- ___*global*___ : Placé devant une fonction, il indique que celle-ci est un *kernel*, c'est à dire une fonction qui va être appelée par le CPU et exécutée par le GPU.
- ___*device*___ (le GPU) : Désigne une fonction qui sera exécutée par le GPU mais qui ne peut être appelée que depuis le GPU (autrement dit depuis une autre fonction ___*device*___ ou depuis une fonction ___*global*___).
- ___*host*___ (le CPU) : Optionnel, il désigne une fonction appelée par le CPU et exécutée sur le CPU, autrement dit une fonction traditionnelle.

En programmation GPU, le processeur central CPU est considéré comme étant l'hôte et le GPU comme un *accélérateur*. Lorsque nous voulons porter une portion de code sur le GPU, il faut créer un *kernel*. La spécificité de ces *kernels* est qu'ils sont exécutés N fois en parallèle par N tâches, appelées *threads*. Les deux lignes suivantes montrent la déclaration puis l'appel de la fonction qui est exécutée sur le GPU avec trois entrées :

```
__global__ void monKernel(parametre1, tableau1, tableau2);
monKernel <<<DimGrid, DimBlock>>>(parametre1, tableau1, tableau2);
```

Les paramètres `DimGrid` et `DimBlock` passés au *kernel* sont des extensions CUDA spécifiques à l'utilisation du GPU. Elles définissent la hiérarchie des *threads*. Les *threads* sont rangés dans des *blocs* qui eux-mêmes sont rangés dans des *grilles*. Ces paramètres sont les suivants :

- `DimGrid` : C'est la dimension de *grille*. C'est-à-dire le nombre de *blocs* que l'on peut mettre dans une grille. La grille peut être de dimensions 1 ou 2.
- `DimBlock` : C'est la dimension de *bloc*. C'est-à-dire, le nombre de *threads* que l'on peut mettre dans un *bloc*. Un *bloc* peut être de dimensions 1, 2 ou 3.

Les *threads* à l'intérieur des *blocs* peuvent communiquer via la mémoire partagée. Des barrières de synchronisation peuvent être imposées par *bloc*. Tous les *threads* d'un bloc sont traités par un multiprocesseur GPU. Pour développer un *kernel*, il faut gérer les notions de dimension de *grille* et de *bloc* car ce sont elles qui déterminent combien de fois sera exécuté le *kernel* en parallèle. Il faut toujours créer un index de *thread* pour que le *kernel* soit exécutable. Pour cela, la compréhension et l'usage de certains indices sont indispensables.

L'indice `threadIdx` est le compteur de **thread**. Dans un *kernel*, il va s'incrémenter jusqu'à atteindre la taille limite d'un *bloc* donné. Comme nous l'avons dit précédemment, la taille d'un *bloc* (`DimBlock`) peut avoir jusqu'à 2 dimensions. L'indice `threadIdx`, étant le compteur de *thread*, peut lui en avoir 3. C'est pourquoi il existe `threadIdx.x`, `threadIdx.y` et `threadIdx.z`.

L'indice `blockIdx` est le compteur de *bloc*. Il fonctionne de la même manière que `threadIdx`. Il s'incrémente jusqu'à atteindre la taille limite d'une *grille* donnée et peut être à 2 dimensions (`blockIdx.x`, `blockIdx.y`).

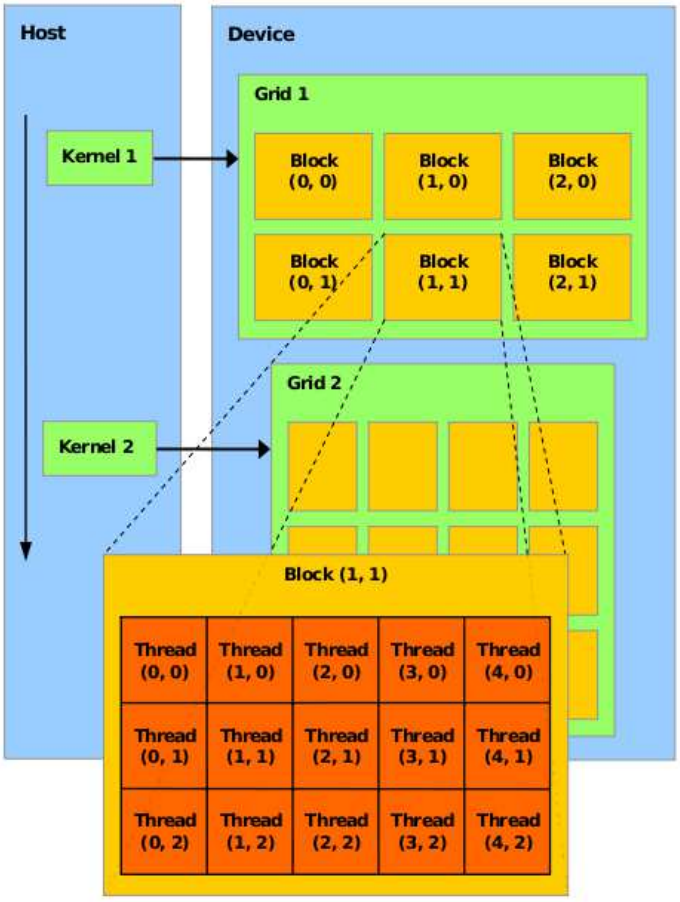


FIGURE 2.7 – Structure de répartition des tâches avec CUDA

2.3.2 Gestion de la mémoire

Une fonction ou *kernel*, s'exécutant sur le GPU, n'a pas accès à la mémoire DRAM centrale de l'ordinateur. Il existe donc une multitude de fonctions prédéfinies pour la gestion de la mémoire sur le GPU. Les principales fonctions sont celles qui se chargent du transfert des données entre la mémoire centrale dite *host*, et la mémoire DRAM qui se trouve sur la carte graphique dite *device* :

- `cudaMalloc` : Alloue de la mémoire globale DRAM du GPU. Elle a la même utilité que la fonction classique `malloc` sur CPU.
- `cudaFree` : Libère la mémoire globale DRAM du GPU. Elle a la même utilité que la fonction classique `free` sur CPU.
- `cudaMemcpy` : Copie des données (3 options) :
  - `cudaMemcpyHostToDevice` : Entre la mémoire centrale CPU et la mémoire globale du GPU.
  - `cudaMemcpyDeviceToHost` : Entre la mémoire globale du GPU et la mémoire centrale CPU.
  - `cudaMemcpyDeviceToDevice` : Entre deux variables sur le GPU.

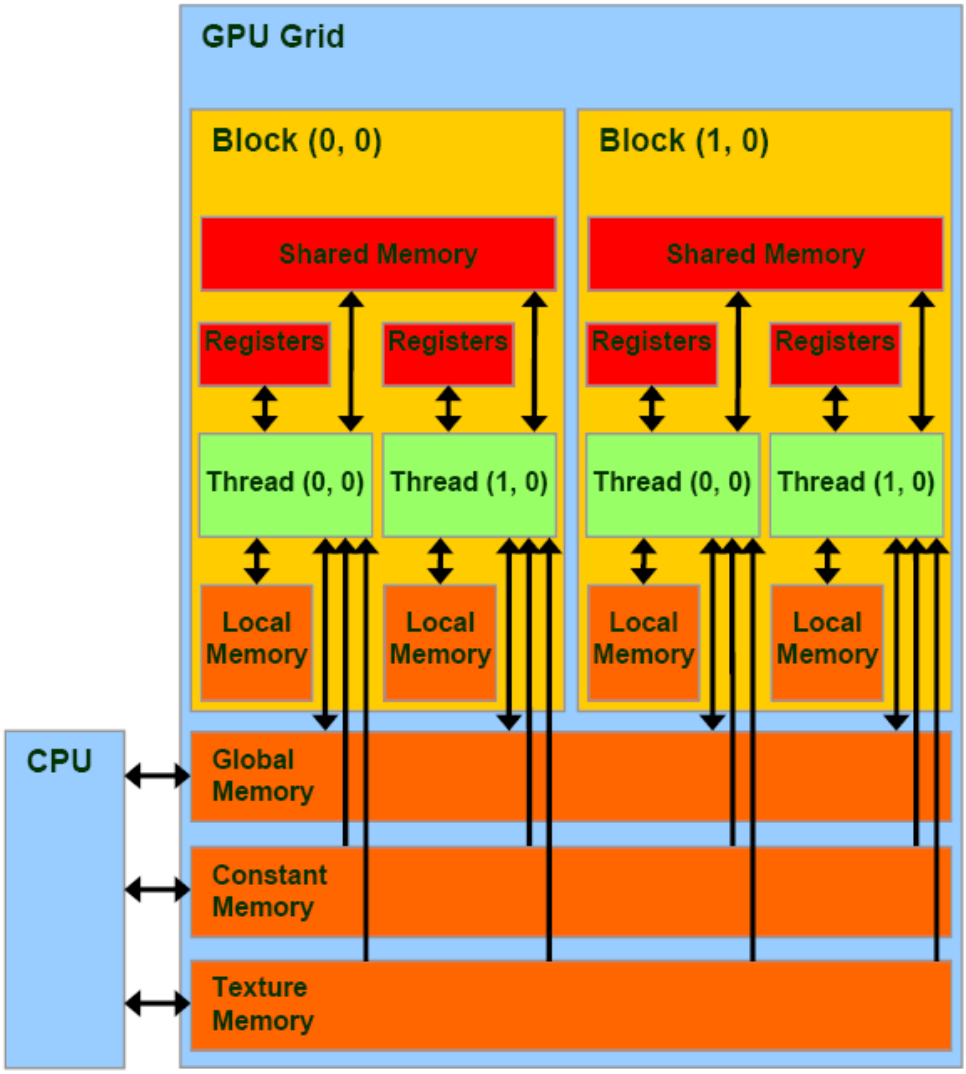


FIGURE 2.8 – Organisation de la mémoire en programmation CUDA. Source : NVIDIA.

Enfin, nous ne rentrerons pas dans le détail, mais il existe des extensions visant à déclarer des variables de mémoire constante, mémoire partagée et mémoire de texture permettant d'utiliser les mémoires caches à accès très rapide décrites dans la Section 2.2.1.

2.3.3 Exemple simple : parallélisation sur CPU et sur GPU

Nous comparons ici un exemple très simple de programmation entre CPU et GPU : l'addition de deux vecteurs [79].

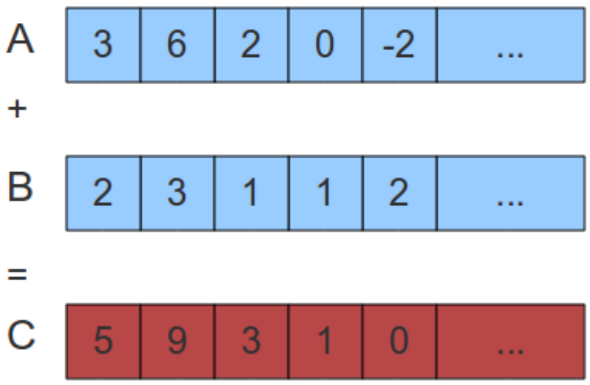


FIGURE 2.9 – Addition de deux vecteurs terme à terme

Sur CPU

En programmation séquentielle classique, il est très facile d'écrire cette fonction :

```
#define N 10000

void add(int *a, int *b, int *c){

  for ( int indice = 0; indice < N; indice = indice + 1 )
  {
  c[indice] = a[indice] + b[indice] ;
  }
}
```

La somme est calculée dans une boucle dont l'indice varie de 0 à $N-1$. A chaque itération, elle additionne les éléments correspondants des vecteurs `a` et `b` et stocke le résultat dans le vecteur `c`.

Si nous disposons de plusieurs CPUs, ou d'un CPU multicœurs, il est très facile de paralléliser cette fonction. Sur un bicœur, nous pouvons soit utiliser une incrémentation de 2 et initialiser la boucle à 0 sur l'un et à 1 sur l'autre, soit traiter les éléments de 0 à $N/2$ sur un cœur et de $(N/2)+1$ à $N-1$ sur l'autre si l'on veut conserver un accès mémoire coalescent [80]. Dans les deux cas, chaque tâche sera le calcul séquentiel de la moitié des éléments des vecteurs. La programmation d'une telle parallelisation est très facile en utilisant la librairie OpenMP (voir Annexe C.2.2).

Sur GPU

Nous pouvons réaliser la même opération sur le GPU en écrivant la fonction `add()` comme une fonction d'accélérateur.

```
#define N 10000

__global__  void add(int *a, int *b, int *c) {

int indice = blockIdx.x;

if (indice < N)
        c[indice] = a[indice] + b[indice];

}

add<<<N,1>>>( a, b, c);
```

La variable `blockIdx.x` est prédéfinie dans CUDA. Nous indiquons à la fonction que nous voulons une *grille* de N *blocs*. Chaque copie de la fonction (*bloc*) aura une valeur de `blockIdx.x` différente allant de 0 à $N-1$. Chaque copie de la fonction s'exécutera en parallèle sur le GPU, mais comme nous l'avons vu, un bloc ne peut s'exécuter que sur un multiprocesseur GPU. Il est dommage d'utiliser un multiprocesseur simplement pour calculer un seul indice du vecteur, puisque nous avons spécifié la valeur 1 à la dimension du *bloc*, c'est à dire qu'il n'y a qu'un seul *thread* par *bloc*. Nous changeons maintenant l'écriture du programme qui lançait N *blocs* de 1 *threads* pour qu'il exécute 1 *bloc* de N *threads* :

```
#define N 10000

__global__  void add(int *a, int *b, int *c) {

int indice = threadIdx.x;

if (indice < N)
        c[indice] = a[indice] + b[indice];

}

add<<<1,N>>>( a, b, c);
```

En plus de changer les paramètres d'exécution du *kernel*, nous avons changé l'indice des données en remplaçant `blockIdx.x` par `threadIdx.x`. Dorénavant, le programmeur va associer les N fonctions à N *threads* au sein d'un seul *bloc*. Comme nous l'avons dit précédemment, les *threads* d'un même *bloc* se s'exécutent que sur un seul multiprocesseur. De plus le nombre de *threads* par *bloc* est limité à 1024 sur un GPU d'architecture Fermi et le nombre de *blocs* par grille est limité à 65535 pour des raisons matérielles. La solution

pour outrepasser cette limite et pour utiliser toutes les unités de calcul du GPU est d'utiliser une combinaison de *threads* et de *blocs*. Le code précédent se transforme en :

```
#define N 10000

__global__ void add(int *a, int *b, int *c) {

int indice = threadIdx.x + blockIdx.x * blockDim.x;

if (indice < N)
        c[indice] = a[indice] + b[indice];

}

add<<<(N+255)/256,256>>>( a, b, c);
```

Le terme $(N+255)/256$ sert à arrondir la division entière de N par 256 à l'entier supérieur (si par exemple $N = 255$, la division entière $255/256$ vaudrait 0). Avec cette arrondi, nous lancerons trop de *threads* mais ils ne s'exécuteront en fait pas à cause de l'instruction `if (indice < N)`.

La programmation de cet exemple simple d'addition de vecteurs nous montre la différence entre une parallélisation sur CPU et une parallélisation sur GPU. Les tâches ou *threads* GPU sont très légères, la parallélisation se fait à très bas niveau. Là où sur GPU une tâche est associée à une seule addition, sur CPU elle est composée de plusieurs additions séquentielles : la parallélisation se fait à haut niveau. Il n'y a théoriquement aucune limite au nombre d'opérations que peut effectuer une tâche CPU.

L'environnement CUDA inclut des libraires de calcul standard qui sont utilisées de la même manière que les fonctions, c'est à dire que la parallélisation est faite à l'intérieur de ces fonctions. Cela pose un problème dans le cas où nous travaillons avec beaucoup de petits paquets de données. Si nous lançons séquentiellement plusieurs fonctions opérant sur un faible nombre de données, le GPU ne sera pas pleinement exploité et les gains de performances seront très limités. Deux solutions s'offrent alors à nous. La première est de ne pas utiliser la fonction de la librairie et reprogrammer la fonction en la parallélisant à deux niveaux, ce qui peut s'avérer compliqué selon l'algorithme que nous voulons implémenter. La deuxième solution consiste à lancer plusieurs fonctions "en concurrence" à l'aide de flux de calcul.

2.3.4 Flux de calcul

Nous avons vu dans la section précédente que la programmation GPU nécessite un parallélisme à bas niveau, dit parallélisme de données. Cependant, les GPUs peuvent exploiter un autre type de parallélisme, semblable à celui que nous pouvons trouver dans les applications *multithreads* d'un CPU. Au lieu d'appliquer simultanément les mêmes opérations sur un grand nombre d'éléments, comme dans le parallélisme de données, le parallélisme de tâche consiste à exécuter en parallèle deux tâches ou plus qui peuvent être totalement différentes.

Les flux CUDA peuvent jouer un rôle important dans l'amélioration des performances des applications. Un flux représente une file d'opérations qui seront exécutées par le GPU dans un certain ordre : ils peuvent contenir des opérations comme des lancements de fonctions, des copies mémoires, des lancements et des arrêts d'évènements. L'ordre de ces ajouts au flux détermine celui dans lequel ces opérations seront exécutées. Les flux peuvent être considérés comme une tâche du GPU et ces tâches peuvent être exécutées en parallèle.

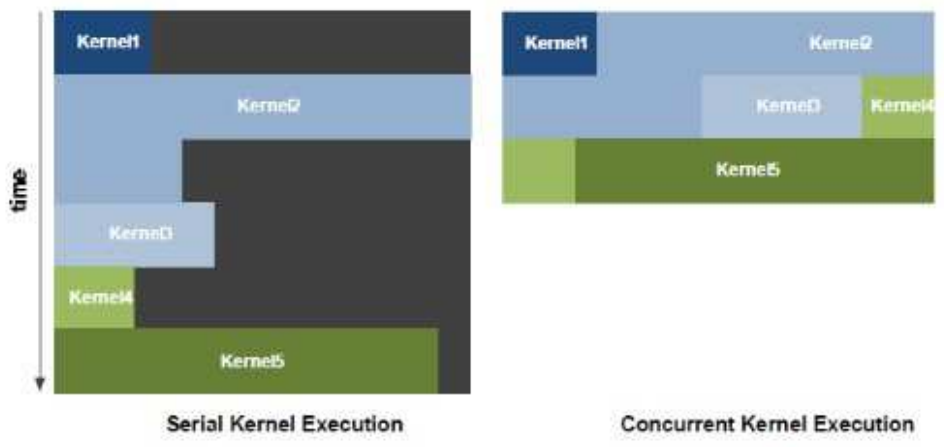


FIGURE 2.10 – *Kernels* lancés séquentiellement (gauche) et en concurrence (droite)

Les flux peuvent avoir deux utilités. Ils permettent le recouvrement des temps de transfert des données, c'est à dire qu'ils permettent de lancer des calculs sur un set de données pendant que d'autres données sont transférées de la mémoire centrale vers la mémoire du GPU et pendant qu'un autre set de données est transféré de la mémoire du GPU vers la mémoire du CPU. Ils permettent aussi de lancer plusieurs fonctions en concurrence, c'est à dire que si la charge du GPU le permet, elles seront traitées en parallèle ce qui est particulièrement intéressant lorsque l'on travaille sur des petits jeux de données (voir Figure 2.10).

2.4 Conclusion

Dans ce chapitre, nous avons brièvement introduit ce que sont les processeurs graphiques GPU. Nous avons vu que leurs caractéristiques les rendent très intéressants du point de vue de la consommation énergétique, du volume occupé et du coût pour de nombreuses applications de calcul scientifique qui sont massivement parallèles.

Les traitements STAP sont facilement parallélisables sur CPU, puisque le traitement est plus ou moins indépendant entre les différentes cases distance, la taille des données à l'intérieur de ces cases distance est relativement petite. L'étude de l'architecture et du langage de programmation des GPUs nous montrent que ces derniers devraient s'avérer efficaces pour effectuer des traitements STAP à condition d'envisager une parallélisation correctement adaptée des algorithmes.

3

Algorithmes et implémentation

Sommaire

L'augmentation de la complexité des systèmes radar ainsi que le traitement du signal associé impliquent une hausse significative de sa charge calculatoire. L'un des obstacles à l'utilisation de traitements STAP est d'ordre technologique, et réside dans l'implémentation physique des algorithmes, liée, comme nous l'avons vu, à l'importante charge de calcul nécessaire. Dans le chapitre précédent, nous avons vu que les GPU offrent potentiellement des puissances de calcul bien supérieures à des processeurs généralistes pour un encombrement et un coût réduits. Cependant l'architecture algorithmique doit pouvoir s'adapter à la parallélisation nécessaire au GPU. Dans ce chapitre, nous développons des algorithmes permettant d'implémenter un traitement STAP classique sur GPU. Nous étudions leurs performances et les comparons à une implémentation traditionnelle sur un processeur généraliste.

3.1 Gestion des données et puissance de calcul pratique

3.1.1 Transfert des données entre mémoire centrale et GPU

Dans le chapitre précédent, nous avons vu que le pont CPU-GPU peut représenter un goulet d'étranglement à cause de son débit limité. Nous allons mesurer les temps de transfert des données entre la mémoire du CPU et la mémoire du GPU. Les paramètres STAP utilisés sont ceux décrits dans le chapitre 1 mais nous prendrons ici un cas "mixte" c'est à dire que nous prenons certains paramètres de la configuration GMTI et certains paramètres de la configuration air-air. Plus précisément, nous simulons un radar comportant $N = 8$ sous-réseaux de réception. Le nombre de cases distance à traiter est $l = 1000$, le nombre d'impulsions $M_p = 128$ pour une PRF de $f_r = 2kHz$ et la fenêtre temporelle choisie est $M = 8$, ce qui implique un temps de traitement total inférieur à $M_p/f_r = 64ms$.

Le cube de données radar est de dimension $8 \times 128 \times 1000$ ce qui représente un volume de données de 8,192 Mo en nombres complexes simple précision (32bits). Nous mesurons le temps de transfert du cube de données qui se trouve dans la mémoire centrale et qui est envoyé vers la mémoire du GPU, et d'autre part, le temps de transfert d'une image Doppler-distance, qui est le résultat en sortie d'un traitement, vers le CPU. Pour le moment, nous ne nous intéressons qu'aux transferts et dans les deux cas deux ensembles de données sont générées aléatoirement. Un descriptif du PC sur lequel sont effectuées les mesures est disponible en Annexe C.3.

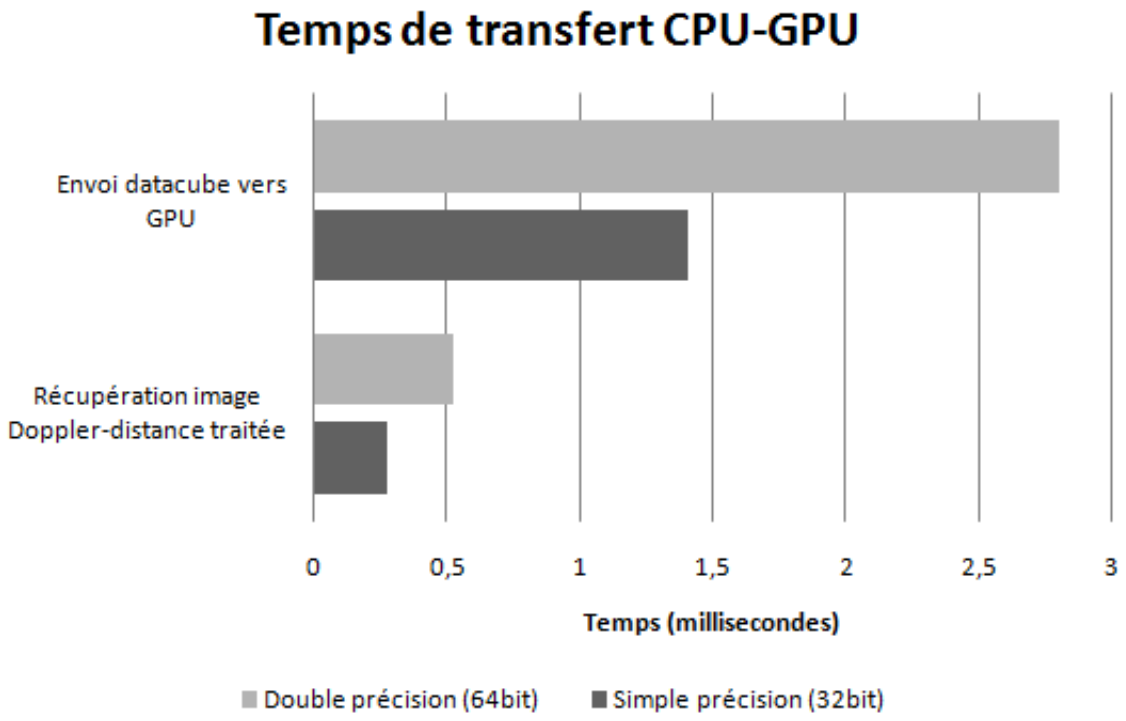


FIGURE 3.1 – Temps de transfert des données radar entre CPU et GPU

La Figure 3.1 montre que le transfert des données vers le GPU se fait à une vitesse d'environ 5,8 Go/s soit 73% du débit théorique maximal du port PCI Express 2.0 qui est de 8 Go/s. Dans le sens inverse, le transfert d'une image Doppler-distance de taille 128×1000 se fait à environ 3,9 Go/s soit 49% de la bande passante théoriquement atteignable. Autant en simple précision qu'en double précision, les temps de transfert sont faibles devant la durée de la rafale entière (64 ms).

3.1.2 Puissance de calcul en pratique

Avant d'implémenter les algorithmes de traitement, nous mesurons en pratique les puissances de calcul maximum atteignables par le CPU et le GPU de notre système. Pour cela, nous utilisons la routine *gemm* qui calcule le produit matriciel de deux matrices. Pour le CPU, nous utilisons la librairie mathématique Intel MKL tandis que pour le GPU nous utilisons la librairie CUBLAS. Cet algorithme, fortement parallèle, est particulièrement bien adapté pour mesurer la puissance de calcul maximum atteignable par un processeur ou un cluster de calcul en pratique [81]. Nous générons aléatoirement deux matrices complexes $\mathbf{A}$ et $\mathbf{B}$ puis nous mesurons le temps mis pour le calcul de $\mathbf{C} = \mathbf{AB}$. Nous divisons ensuite ce temps par la charge de calcul associée ($8N^3$) pour obtenir la puissance de calcul.

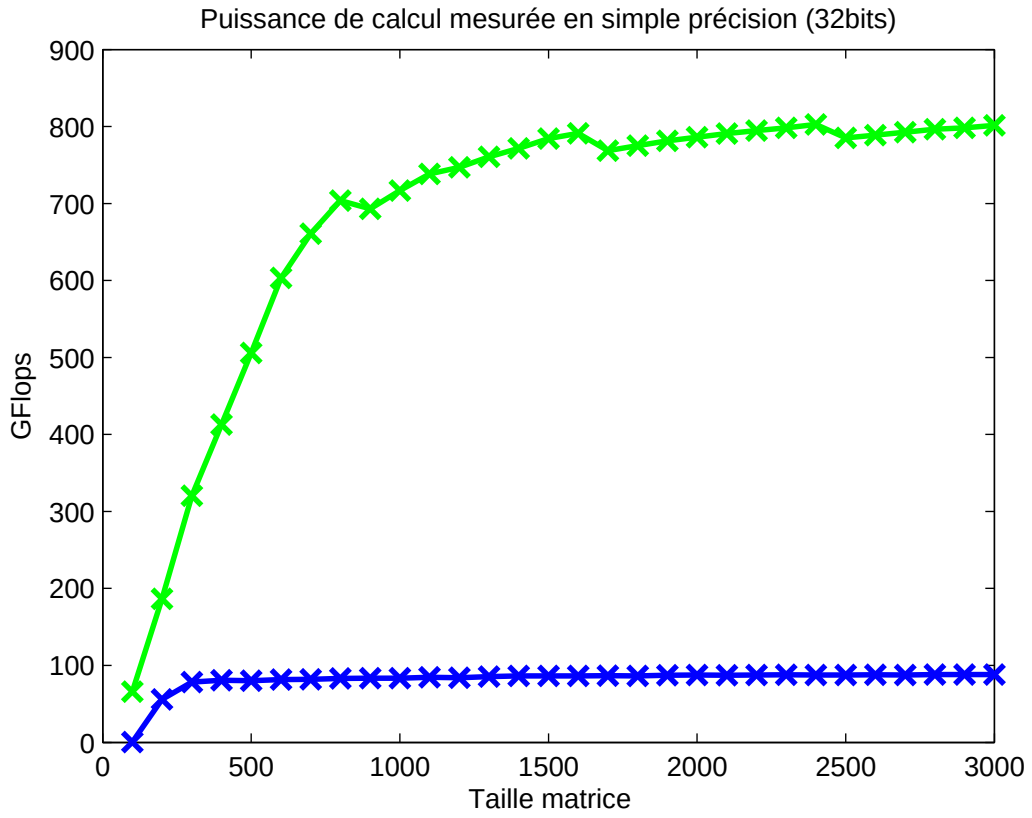


FIGURE 3.2 – Puissance de calcul mesurée pour le CPU (bleu) et sur le GPU (vert)

Nous voyons que les puissances maximales des deux processeurs sont atteintes pour des tailles de matrices différentes, 300×300 pour le CPU et 800×800 pour le GPU. Nous en revenons au problème de parallélisation que nous avons vu au chapitre précédent. Dans cet exemple, la parallélisation est faite à bas niveau, c'est à dire que le calcul d'un élément de la matrice résultante $\mathbf{C}$ est affecté à une tâche. Comme il faut beaucoup de tâches pour exploiter le GPU, les GPUs sont efficaces à partir d'une certaine taille de matrice. En STAP, nous travaillons sur des matrices de covariance dont la taille est souvent inférieure à 100×100. Comme nous travaillons sur un grand nombre de matrices, nous arriverons facilement à exploiter la puissance du CPU puisqu'il est très facile d'affecter le calcul d'une matrice à une tâche CPU, chaque matrice étant quant à elle calculée séquentiellement. Le GPU en revanche a lui besoin d'un grand nombre de tâches légères, il faudra pour pleinement utiliser sa puissance deux niveaux de parallélisation : à bas niveau, c'est à dire au niveau de chaque élément de matrice, et à haut niveau, c'est à dire pour chaque matrice.

La figure suivante donne les valeurs pour des calculs effectués en simple précision. A partir des spécifications des processeurs (voir Annexe C.2.1 et C.3.2), nous pouvons comparer ces résultats aux valeurs théoriques :

	CPU (Intel X5570)	GPU (Tesla C2050)
Puissance théorique (GFlop/s)	93.7	1030
Puissance mesurée (GFlop/s)	88.2	801.4
Ratio mesurée/theorique	94 %	78 %

En pratique, malgré un rapport puissance pratique sur puissance théorique légèrement inférieur au CPU, le GPU affiche une puissance de calcul plus de 9 fois supérieure au CPU.

3.2 Traitement STAP : algorithmes et performances sur GPU

Dans cette Section, nous décrivons l'implémentation des différentes étapes d'un traitement STAP classique. Comme nous l'avons vu dans le Chapitre 1, ce traitement STAP peut se décomposer en 4 étapes.

3.2.1 Estimation des matrices de covariance

Cette étape du traitement vise à calculer les matrices de covariance spatio-temporelles pour toutes les cases distances et à appliquer un *diagonal loading* (voir Annexe B.4). Cela revient à effectuer, pour chaque case distance les opérations suivantes :

$$\mathbf{R_k} = \frac{1}{K_t} \sum_{i=1}^{K_t} \mathbf{x_i} \mathbf{x_i^H} + \delta \mathbf{R_{th}}$$

Le vecteur $\mathbf{x_i}$ est de taille $N \times M$. Pour calculer efficacement $\mathbf{R_k}$, nous ré-organisons les données selon le formalisme défini en Partie 1.

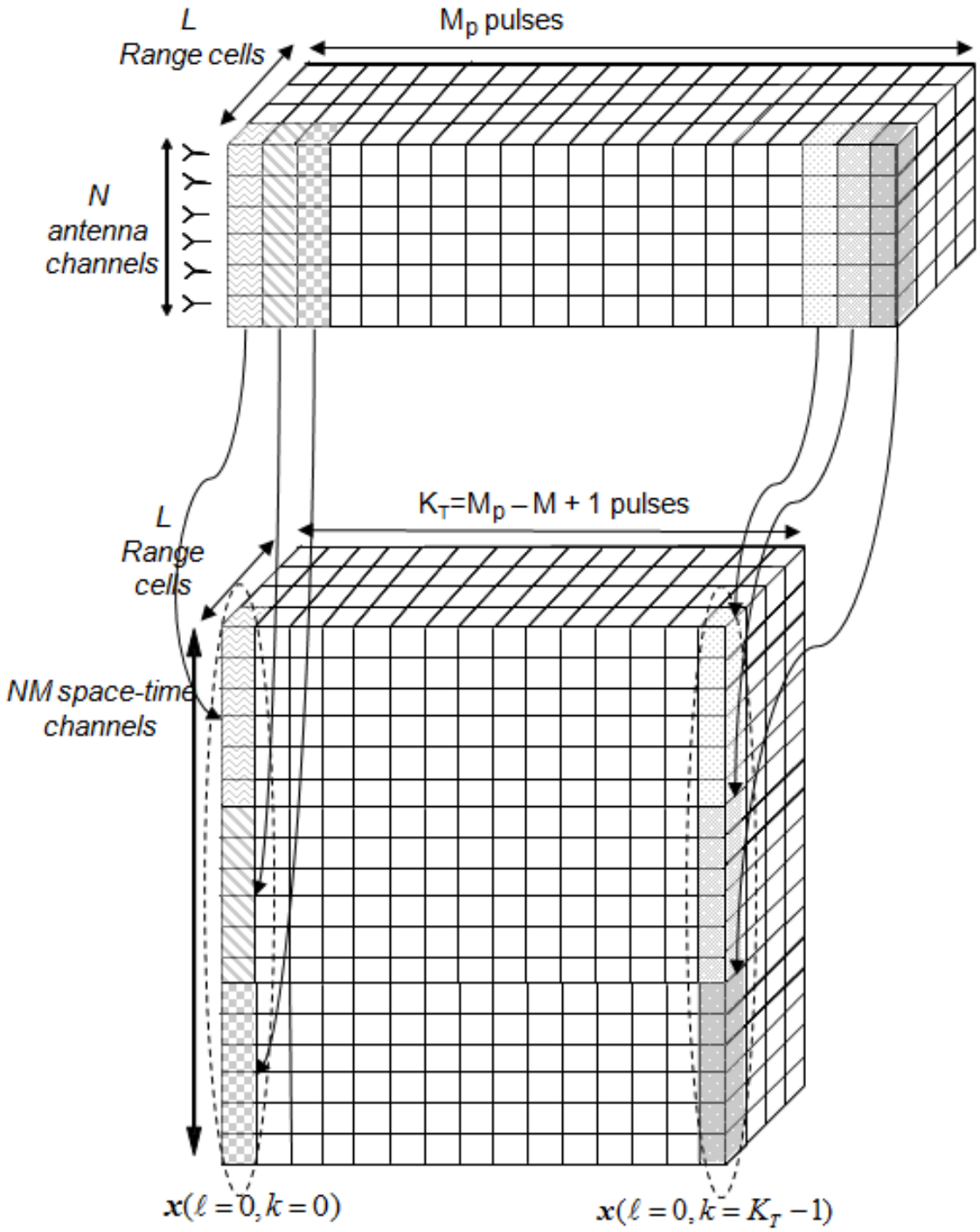


FIGURE 3.3 – Réorganisation des données pour l'estimation des matrices de covariance. Cette réorganisation suit le formalisme vu dans (2.9)

De cette façon, pour chaque chaque case distance la matrice de données de taille $N \times M_p$ devient une matrice X de taille $NM \times K_t$. Une fois ré-organisées, les matrices de covariance sont calculées par multiplication matricielle :

$$\mathbf{R_k} = \frac{1}{K_t}\mathbf{X}\mathbf{X^H} + \delta\mathbf{R_{th}}$$

Sur GPU, nous comparons trois méthodes pour effectuer cette étape. La plus simple est d'utiliser simplement la fonction *gemm* incluse dans CUDA, les matrices sont calculées à la suite. D'après ce que nous avons vu dans la section précédente, les matrices, de taille 64×121 sont trop petites pour exploiter pleinement le GPU et nous nous attendons donc à une faible efficacité. Cette méthode servira de référence.

Multiplication matricielle par *batch* sur GPU

Dans un deuxième temps, nous programmons une fonction dérivée de celle décrite dans [82]. La différence vient du fait que les matrices sont traitées par *batch*, c'est à dire que plusieurs multiplications de matrices se font en même temps (voir Annexe C.4). Chaque élément de $\mathbf{R_k}$ est calculé de la façon suivante :

$$\mathbf{R_k}(i,j) = \mathbf{X}(i,1)\mathbf{X^H}(1,j) + \mathbf{X}(i,2)\mathbf{X^H}(2,j) + ... + \mathbf{X}(i,K_t)\mathbf{X^H}(K_t,j) \tag{3.1}$$

Chaque *thread* est affecté au calcul d'un élément de matrice et calcule le résultat de (3.1).

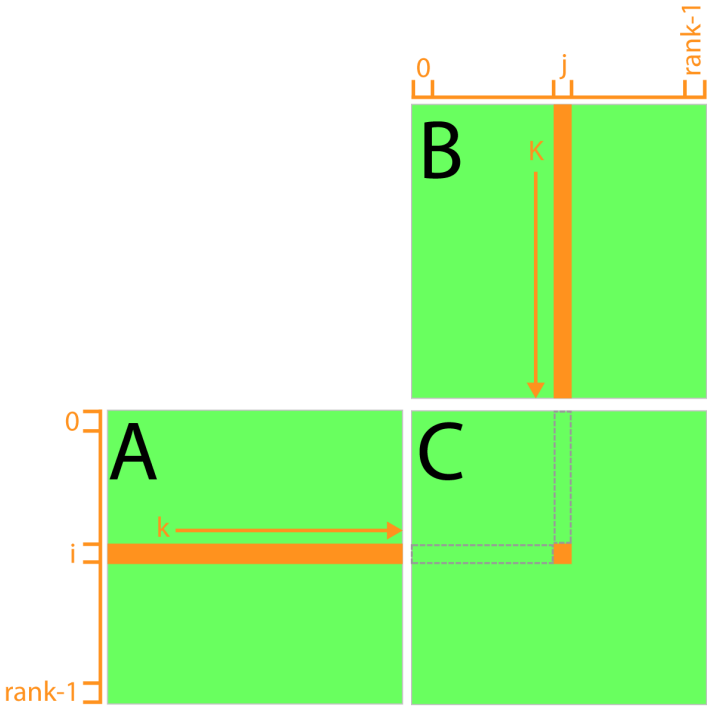


FIGURE 3.4 – Multiplication de matrices sur GPU, chaque *thread* traite un élément de matrice. Source : NVIDIA

Nous voyons que chaque *thread* charge $2K_t$ éléments pour calculer une valeur de $\mathbf{R_k}$. Comme l'accès à la mémoire globale est relativement lent, les *threads* passeront la majeure partie du temps à attendre les données et l'exécution ne sera pas efficace. Pour réduire le nombre d'accès mémoire, nous utilisons la méthode détaillée dans [83], et nous allons pré-charger une partie des matrices dans la mémoire partagée (voir Section 2.3.2), d'où les *threads* pourront y accéder beaucoup plus rapidement. Idéalement, nous devrions charger les deux matrices entièrement dans la mémoire partagée. Malheureusement, chaque multiprocesseur contient seulement 48 Ko de mémoire. Nous allons donc découper les matrices en portions plus petites, de taille BLOCK_SIZE×BLOCK_SIZE comme indiqué sur la Figure 3.5.

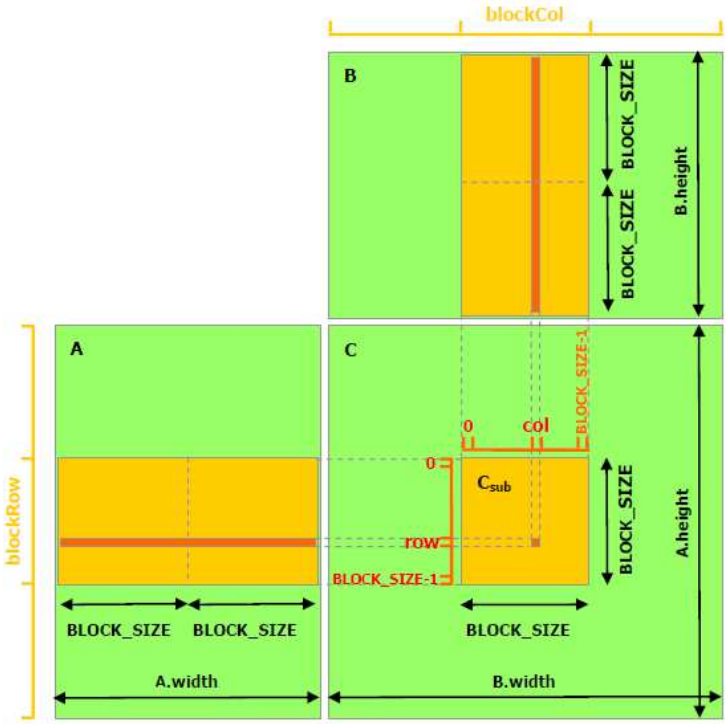


FIGURE 3.5 – Multiplication de matrices sur GPU par *blocs*. Chaque *bloc* est stocké en mémoire partagée. Source : NVIDIA

Lorsque chaque *thread* d'un *bloc* a terminé le calcul de la somme, la sous-matrice suivante est chargée à partir de $\mathbf{X}$. Nous fixons BLOCK_SIZE×BLOCK_SIZE = 16×16 de manière à ce que les sous-matrices contiennent 256 éléments, c'est à dire que pour le calcul d'une matrice de covariance de taille 64×64 il faudra 4×8 *blocs*. Comme K_t n'est pas un multiple de 8, nous remplissons les matrices de 0 de manière à ce que X soit de taille 64×128. En simple précision, un élément requiert 8 octets, donc un *bloc* requiert $2 \times 8 \times 256$ = 4 Ko, donc 12 *blocs*. Plus il y a de *blocs* qui résident sur le multiprocesseur, plus il y aura de chances que l'ordonnanceur soit capable de trouver des *threads* qui puissent se lancer alors que d'autres sont en attente d'accès mémoire. Les *blocs* qui calculent chaque sous-matrices sont placés sur les deux premières dimensions de la grille et la troisième dimension indexe les *blocs* des différentes matrices (voir Annexe C.4).

Une fonction similaire appelée *gemmBatched* a été ajoutée dans la version 4.1 de CUDA. Nous comparons les performances de notre fonction que nous appellerons *matrixmulbatched* avec la fonction *gemmBatched* pour l'estimation des matrices de covariance. La Figure 3.6 montre que la fonction *gemmBatched* est 37% plus efficace que notre fonction *matrixmulbatched*. Nous utiliserons donc *gemmBatched* dans la suite.

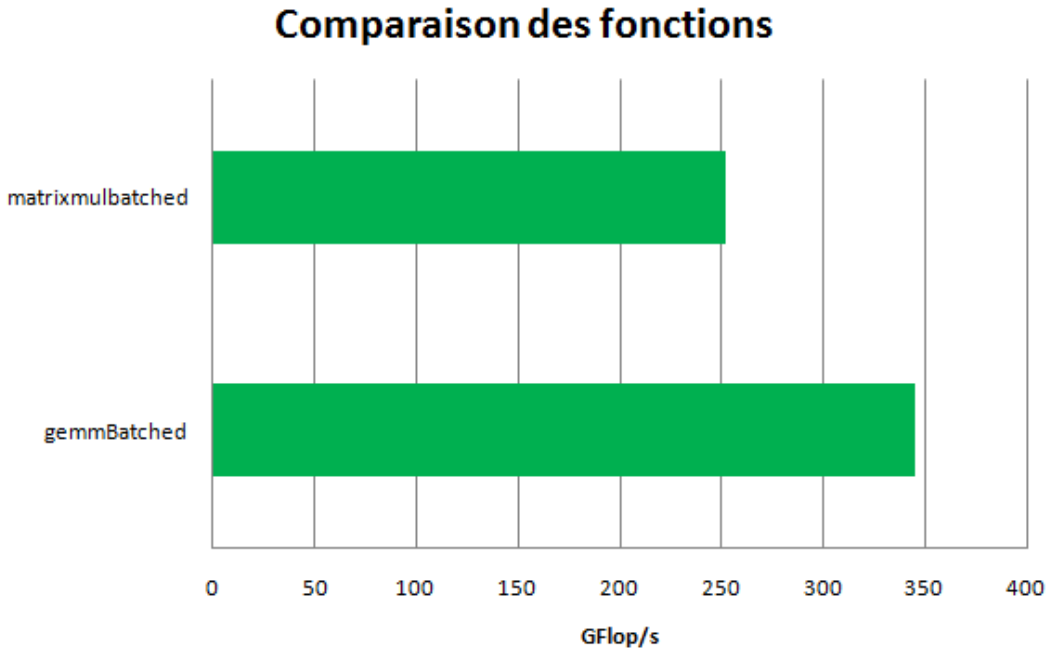


FIGURE 3.6 – Comparaison des efficacités des fonctions *matrixmulbatch* et *gemmBateched* en simple précision

Multiplication matricielle par flux sur GPU

Enfin, une méthode beaucoup plus rapide à programmer consiste à créer 1000 flux de calcul et chaque flux sera associé à une multiplication de matrice effectuée avec la fonction CUDA *gemm*. Les fonctions seront lancées en concurrence (voir Section 2.3.4). Au maximum, une fonction peut être lancée par multiprocesseur. Dans ce cas, plusieurs matrices seront calculées sur les différents multiprocesseurs et les éléments de chaque matrice seront calculés par des unités différentes au sein d'un multiprocesseur.

Multiplication matricielle sur CPU

Sur CPU, nous choisissons d'affecter avec la librairie de programmation parallèle OpenMP (voir Annexe C.2.2), l/N cases distance aux N cœurs du CPU, c'est à dire que chacun des 4 cœurs traite 250 cases distance. Les multiplications de matrices sont effectuées avec la fonction *gemm* de la librairie Intel MKL (mode séquentiel).

Performances

Nous comparons tout d'abord les performances des trois méthodes pour l'estimation des matrices de covariance sur GPU : la méthode de référence *gemm* séquentielle (*gemm* seq.), la méthode par *batchs* (*gemmBatched*) et la méthode par flux (*gemm* flux). Le tableau suivant présente les puissances de calcul des trois méthodes :

Estimation sur GPU :	*gemm* seq.	*gemmBatched*	*gemm* flux
Simple précision (32bit)	23 GFlop/s	336 GFlop/s	345 GFlop/s
Double précision (64bit)	19 GFlop/s	129 GFlop/s	132 GFlop/s

Ces chiffres montrent que les deux approches par *batchs* et par *flux* sont très proches, cette dernière étant légèrement plus rapide et sa programmation est plus simple. Nous garderons cette méthode pour la comparaison avec le CPU.

Le tableau suivant présente les temps de traitement pour l'estimation des matrices de covariance du GPU avec l'implémentation par flux et CPU.

Plateforme :	GPU	CPU	Ratio (GPU/CPU)
Simple précision (32bit)	11.5 ms	72.7 ms	6.34
Double précision (64bit)	20.5 ms	131 ms	5.43

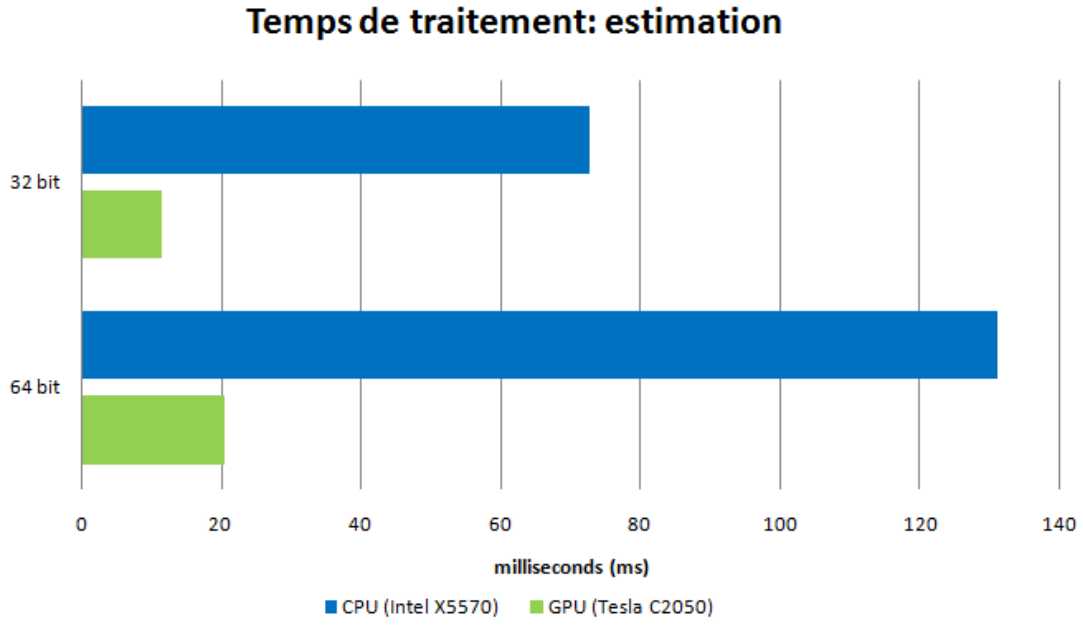


FIGURE 3.7 – Comparaison des temps de traitement entre GPU et CPU pour l'estimation des matrices de covariance

3.2.2 Inversion des matrices de covariance

La deuxième étape du traitement consiste en l'inversion de toutes les matrices de covariance. Classiquement, pour obtenir $\mathbf{R_k^{-1}}$ à partir de $\mathbf{R_k}$ nous calculons la décomposition

LU de la matrice de covariance (nous pouvons utiliser une décomposition de Cholesky si la matrice est hermitienne définie positive comme dans notre cas) puis nous calculons l'inverse de la matrice en résolvant le système $\mathbf{LU}\,\mathbf{X} = \mathbf{I}$. La librairie propriétaire CULA [37] possède deux fonctions permettant d'effectuer ces deux opérations mais il n'est pas possible d'en lancer plusieurs en concurrence et nous aurons le même problème d'efficacité qu'avec la fonction *gemm* utilisée séquentiellement.

Sur CPU, nous allons utiliser les deux fonctions *getrf* qui calcule la décomposition **LU** et *getri* qui trouve l'inverse de la matrice à partir de la décomposition **LU**. Pour notre cas précis, nous aurions pu utiliser les fonctions *potrf* et *potri* qui calculent la décomposition de Cholesky et l'inverse et sont plus rapides, $4(MN)^3$ contre $8(MN)^3$ opérations totales, mais nous utilisons *getrf* et *getri* pour rester dans un cas général.

Sur GPU, nous utilisons une implémentation calculant l'inverse d'une matrice par la méthode du pivot de Gauss [84].

$$\left[\begin{array}{c|ccccc} & 1 & 0 & 0 & 0 & 0 \\ & 0 & 1 & 0 & \vdots & 0 \\ \mathbf{R_k} & 0 & 0 & 1 & 0 & \vdots \\ & 0 & \dots & 0 & \ddots & 0 \\ & 0 & \dots & 0 & 0 & 1 \end{array}\right] \xrightarrow{\text{pivot de Gauss}} \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & & & & & \\ 0 & 1 & 0 & \vdots & 0 & & & & & \\ 0 & 0 & 1 & 0 & \vdots & \mathbf{X_1} & \mathbf{X_2} & \dots & \mathbf{X_{NM}} & \\ 0 & \dots & 0 & \ddots & 0 & & & & & \\ 0 & \dots & 0 & 0 & 1 & & & & & \end{array}\right]$$

et où $\mathbf{R_k^{-1}} = [\mathbf{X_1}\,\mathbf{X_2}\,\dots\,\mathbf{X_{NM}}]$. Chaque *bloc* traite une matrice et cette matrice est stockée dans la mémoire partagée, les inverses des matrices sont donc calculées en parallèle. Le nombre de *blocs* est égal au nombre de matrices à traiter. Chaque *bloc* résout donc un système. Le tableau et la figure suivante présentent les performances sur CPU et sur GPU de cette étape d'inversion de la matrice de covariance.

Plateforme :	GPU	CPU	Ratio (GPU/CPU)
Simple précision (32bit)	14.1 ms	54.4 ms	3.9
Double précision (64bit)	36.0 ms	94.1 ms	2.5

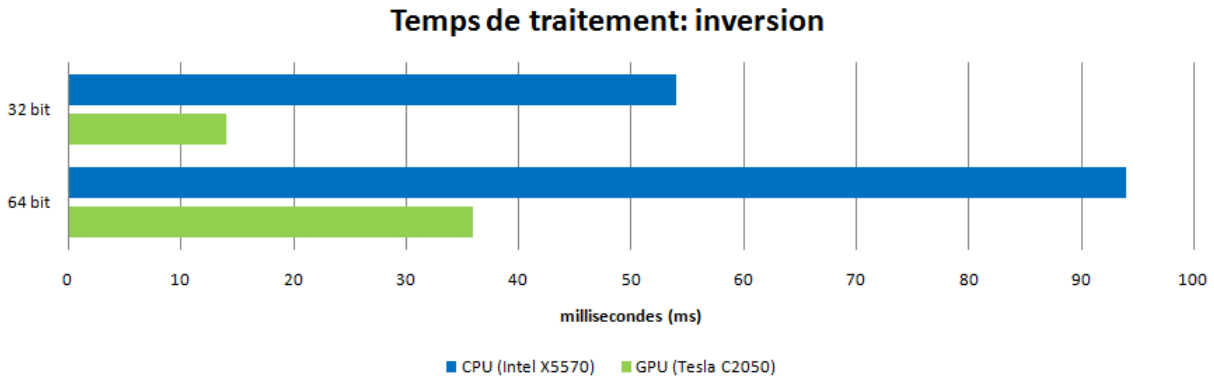


FIGURE 3.8 – Temps de traitement CPU et GPU pour l'inversion

3.2.3 Calcul des poids des filtres STAP

Cette étape a pour but de calculer les filtres $\mathbf{w_k} = \frac{\mathbf{R_k^{-1}s_s}}{\mathbf{s_s^H R_k^{-1} s_s}}$ pour chaque case distance k. L'opération la plus lourde est la multiplication matrice-vecteur $\mathbf{R_k^{-1}s_s}$. L'opération $\mathbf{s_s^H R_k^{-1} s_s}$ est un produit scalaire de vecteurs de taille MN et l'opération finale est la division du vecteur $\mathbf{R_k^{-1}s_s}$ par le scalaire $\mathbf{s_s^H R_k^{-1} s_s}$. Pour l'implémentation CPU et GPU, nous utilisons les fonctions *gemv* et *dotc* des librairies CUDA et MKL qui calculent respectivement le produit matrice-vecteur et le produit scalaire. Pour la division, nous utilisons une fonction très similaire à la fonction d'addition de vecteur vue dans le Chapitre 2.3.3. Sur GPU, nous utilisons des flux pour lancer les fonctions en concurrence. Sur CPU, nous adoptons la même approche que précédemment en parallélisant sur les cases distances comme nous l'avons fait pour l'estimation des matrices de covariance et leurs inversions.

Performances

Le tableau suivant et la figure 3.9 présentent les temps d'exécution du calcul des filtres STAP.

Plateforme :	GPU	CPU	Ratio (GPU/CPU)
Simple précision (32bit)	3.6 ms	3.7 ms	1.03
Double précision (64bit)	4.1 ms	6.2 ms	1.51

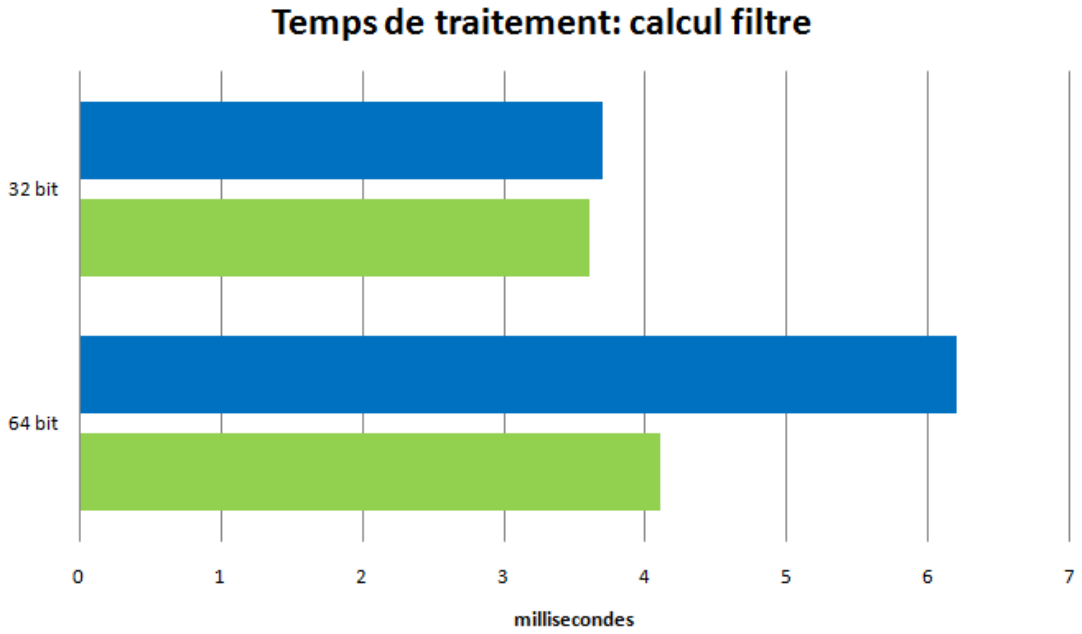


FIGURE 3.9 – Comparaison des temps de traitement entre GPU et CPU pour le calcul des filtres STAP

3.2.4 Application des filtres et transformée de Fourier

La dernière étape du traitement STAP consiste en l'application des filtres $\mathbf{y_k} = \mathbf{w_k^H X_k}$ et passage en plan Doppler-distance par FFT pour détection $\mathbf{y_k(t_n)} \Longrightarrow \mathbf{y_k(f_n)}$. La première opération $\mathbf{y_k} = \mathbf{w_k^H x_k}$ est une multiplication entre le vecteur $\mathbf{w_k^H}$ de taille MN calculé lors de l'étape précédente et la matrice de données $\mathbf{X_k}$ de taille $MN \times K_t$. Pour cette opération, nous utilisons la fonction *gemv* de la même façon que précédemment, c'est à dire en utilisant des flux pour le GPU et une parallélisation par case distance sur CPU. Le calcul de la FFT ne requiert pas d'utilisation de flux parce que CUDA propose nativement des fonctions FFT qui permettent de lancer plusieurs FFTs en parallèle, de la même manière que la fonction *gemmBatched* dans la Section 3.2.1. Les transformées de Fourier sont faites sur les K_t impulsions mais nous appliquons un *zero-padding* pour que la FFT soit faite sur 128 points et non sur 121 points. Nous revenons ainsi au nombre de points Doppler avant traitement STAP et les FFTs sont beaucoup rapides lorsqu'elles sont faites sur des vecteurs dont la taille est une puissance de 2. Avec des vecteurs de taille 128, le GPU atteint une puissance de calcul en pratique d'environ 250 GFlop/s contre 50 GFlop/s pour le CPU (voir Annexe C.7.1).

Performances

Le tableau suivant et la figure 3.10 présentent les temps de calcul pour l'application du filtre STAP et pour les FFTs.

Plateforme :	GPU	CPU	Ratio (GPU/CPU)
Simple précision (32bit)	3.8 ms	3.3 ms	0.89
Double précision (64bit)	3.9 ms	5.3 ms	1.37

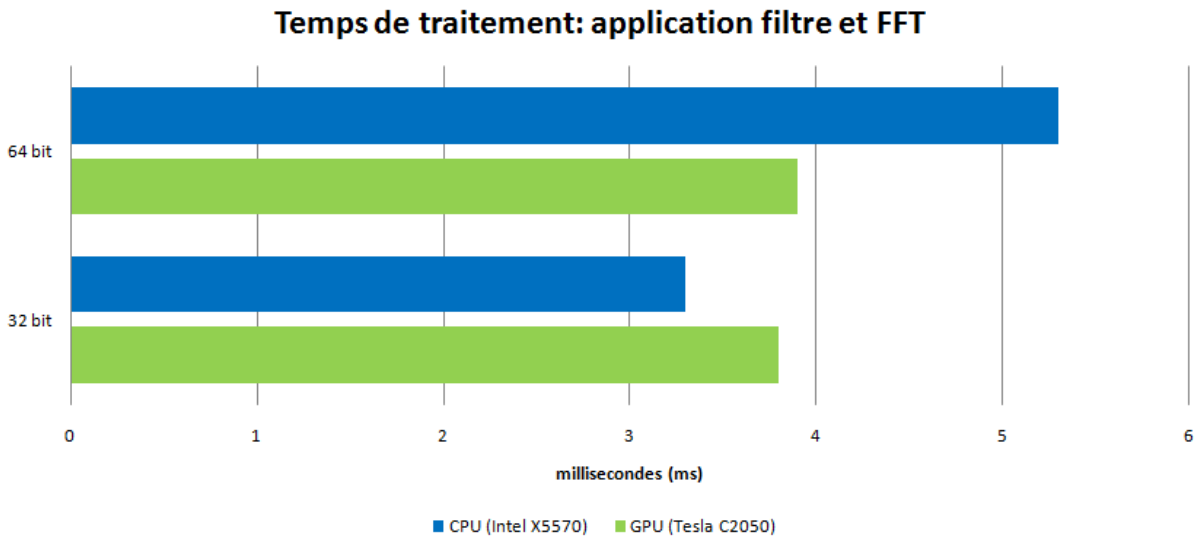


FIGURE 3.10 – Comparaison des temps de traitement entre GPU et CPU pour le l'application des filtres et les FFTs

3.2.5 Récapitulatif des performances

Les figures 3.11 et 3.12 résument les performances que nous avons mesurées pour toutes les étapes du traitement STAP dans les sections précédentes. Comme nous pouvons le voir, le gain qu'apporte le GPU est faible pour les deux étages de traitement léger, c'est à dire le calcul des filtres et l'application des filtres plus transformée de Fourier. Pour ces traitements, le gain de performance qu'amène le GPU ne dépasse pas le facteur 2 et est même légèrement négatif dans un cas en simple précision. Cependant, ces deux étages de traitement ne représentent que moins de 4% de la charge totale d'un traitement STAP classique (voir Chapitre 1.2). Les deux étapes lourdes que sont l'estimation des matrices de covariance et leur inversion montrent quant à elles que le GPU apporte un gain d'un facteur 3,9 à 6,34 en simple précision et un gain d'un facteur 2,5 à 5,43 en double précision. Notons aussi que le gain est supérieur pour l'estimation des matrices de covariance que pour les inversions des matrices. Cela est normal puisque que l'algorithme d'estimation est plus parallèle que celui de l'inversion.

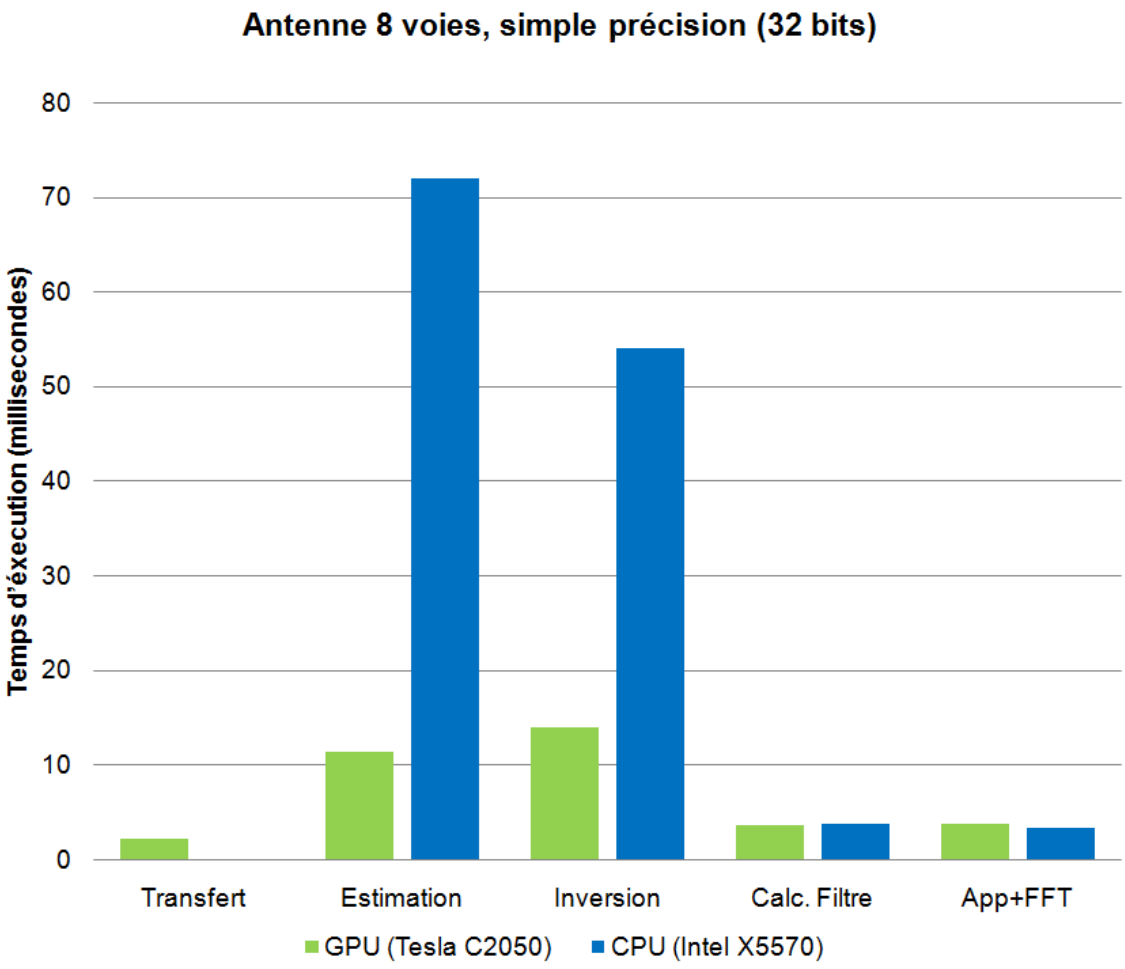


FIGURE 3.11 – Récapitulatif des temps de traitements des étages STAP

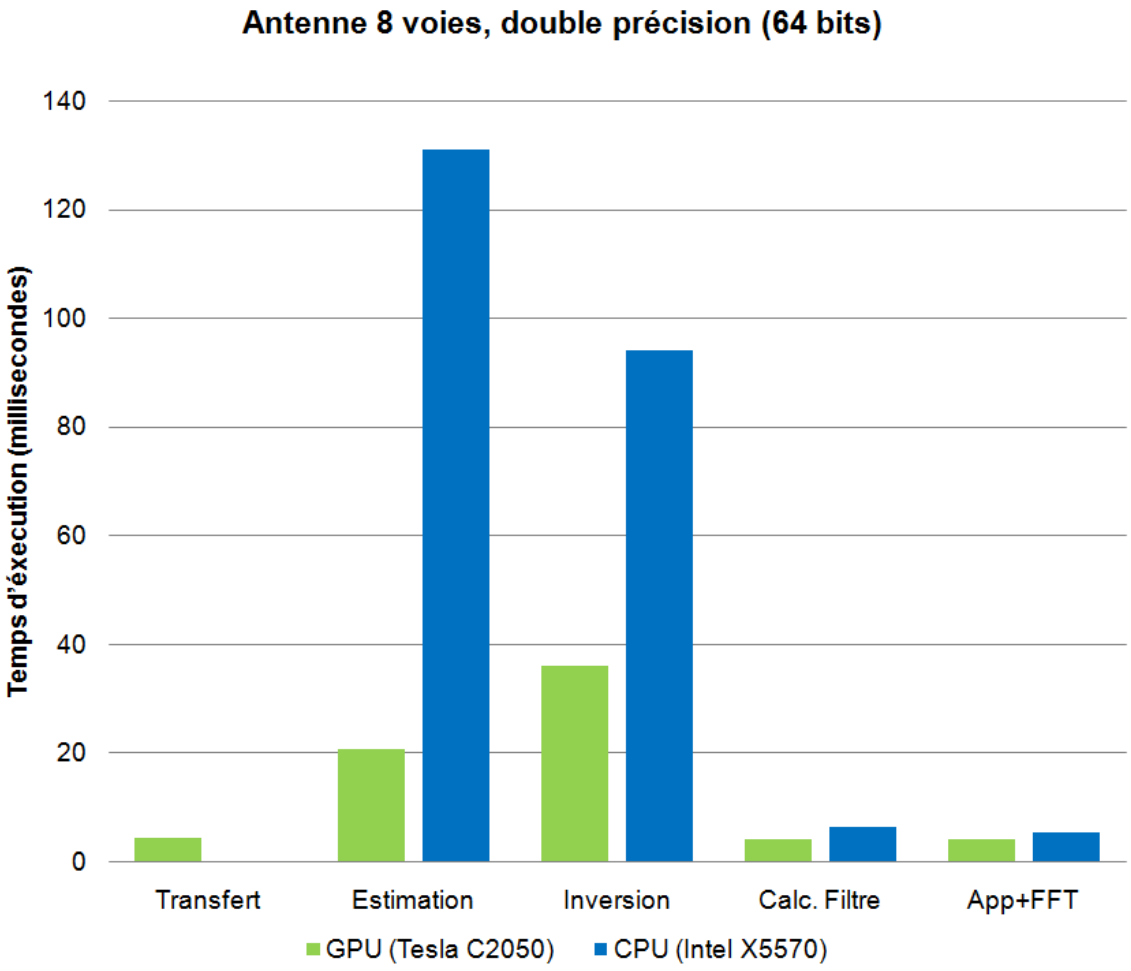


FIGURE 3.12 – Récapitulatif des temps de traitements des étages STAP

3.3 Architecture très efficace ARM-GPU pour les systèmes embarqués

3.3.1 La carte CARMA

Si la motivation pour les radars basés au sol est la réduction du coût de l'achat du système de calcul, pour les radars embarqués nous cherchons à réduire la consommation électrique ainsi que le volume occupé par ce système. Dans ce contexte, l'architecture ARM-GPU proposée par le constructeur de GPU NVIDIA, appelée CARMA est très prometteuse. CARMA est un prototype d'architecture visant à accroitre l'efficacité énergétique des plateformes de calcul GPU. Aujourd'hui, les progrès dans l'amélioration de cette efficacité énergétique se font principalement sur les marchés des smartphones et des tablettes. Les processeurs ARM [85] ont un avantage en terme d'architecture et d'expérience puisque ce sont ces processeurs qui équipent principalement ces produits.

Comme nous l'avons vu dans les chapitres précédents, du fait de leur architecture, les GPUs sont très efficaces pour les calculs massivement parallèles. Par exemple, un GPU NVIDIA de type Fermi a un ratio dissipation thermique à puissance de calcul de 225 pJ/flop, alors qu'un CPU x86 Intel Westmere de même génération est à 1700 pJ/flop [86]. Un GPU ne peut cependant pas exécuter un noyau système seul, et doit être piloté par un CPU. L'efficacité énergétique du système complet sera donc impactée négativement par la présence du CPU pilote. Évidement, plus nous augmenterons le nombre de GPUs (dans la limite de ce qu'un CPU peut gérer), plus on tendra vers l'efficacité énergétique des GPUs. Le but de la carte prototype CARMA est d'explorer la possibilité d'utiliser des GPUs pilotés par un processeur CPU à architecture ARM dans le but d'augmenter l'efficacité des systèmes de calcul.

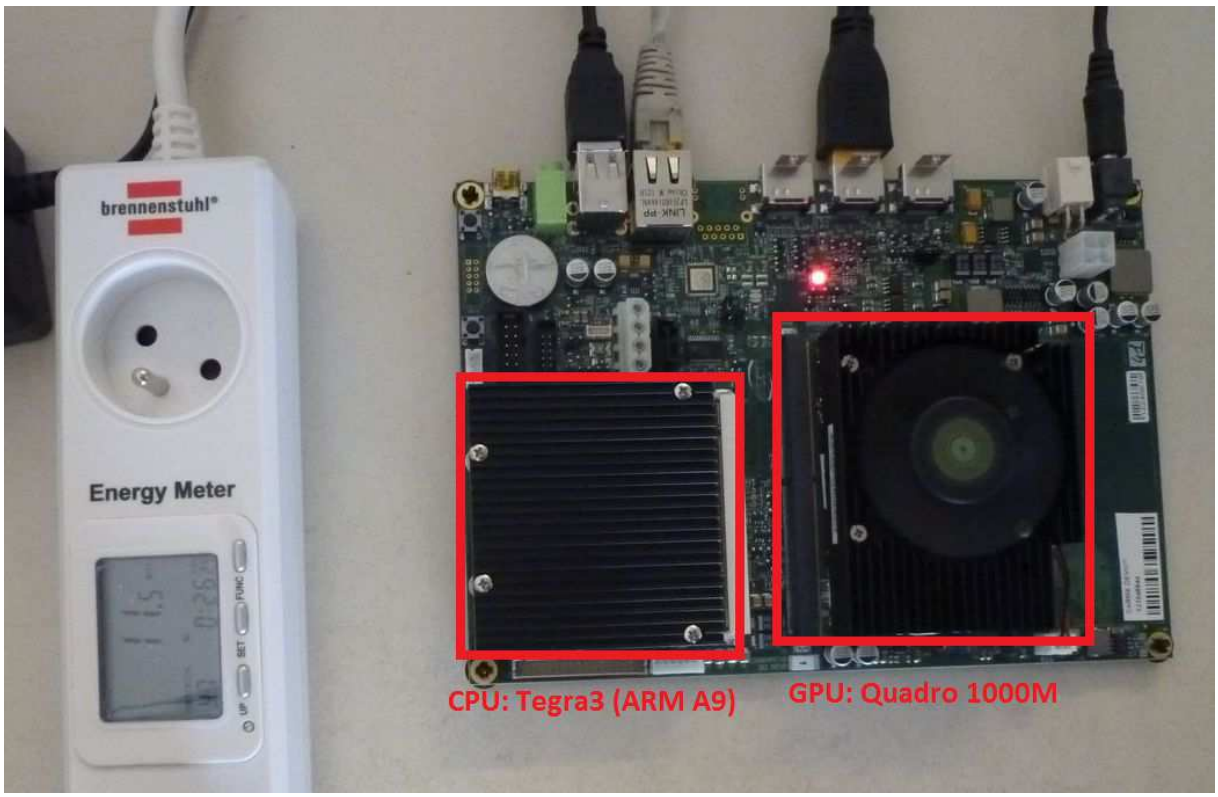


FIGURE 3.13 – Photographie de la carte CARMA. Le CPU est dans le carré gauche, le GPU dans le carré à droite. Le Wattmètre à gauche mesure la consommation de la carte.

Sur la carte CARMA, le processeur traditionnel x86 (Intel ou AMD) est remplacé par un processeur ARM ultra-basse consommation. Ce CPU est un Tegra T30 "Kal-El" ayant quatre cœurs de type ARM Cortex-A9 avec extensions NEON et VFPv3, comme nous pouvons en trouver dans plusieurs modèles de smartphones et de tablettes. Le GPU est quant à lui un NVIDIA Quadro 1000M que l'on peut trouver dans des PC portables professionnels d'entrée de gamme. Ce GPU a une puissance de calcul théorique maximale de 268 GFlop/s, il est de même génération et de même architecture que le GPU présent dans la carte Tesla C2050 qui, lui, a une puissance de calcul crête de 1030 GFlop/s. La carte embarque son propre système d'exploitation basé sur un noyau Linux 3.1.10.

La carte étant un prototype, le compilateur GPU ne supporte pas encore la compilation sur processeur ARM : les programmes doivent être compilés sur un PC standard puis transférés sur la carte. Enfin, le Quadro 1000M étant un GPU d'entrée de gamme, la puissance de calcul en double précision est bridée. Nous testons ici les deux premiers étages du traitement STAP décrits précédemment en simple précision. Ensemble, ces deux étages comprennent plus de 95 % de la charge de calcul du traitement STAP. Pour ces essais, le processeur central x86 est toujours un Intel Xeon X5570, le GPU est une carte NVIDIA Tesla C2050. Nous comparons les vitesses d'exécution du traitement s'exécutant sur ces deux processeurs avec les vitesses d'exécution du traitement sur la carte CARMA.

Zoom sur le GPU

Comme nous l'avons dit, la carte CARMA embarque une carte NVIDIA Quadro 1000M qui est composée d'un GPU GF108 basé sur l'architecture Fermi. Ce GPU possède 96 cœurs CUDA. Nous avons vu dans le Chapitre 2 que le processeur GPU Fermi de base, le GF100, est composé de 16 multiprocesseurs eux-mêmes constitués de 32 unités de calcul, appelées cœurs CUDA. Sur la carte NVIDIA Telsa, deux des multiprocesseurs sont désactivés ce qui explique les 448 coeurs CUDA actifs. Le GPU GF108, lui, ayant 96 cœurs, nous pouvons penser qu'il est fait de 3 multiprocesseurs. C'est en fait un peu plus compliqué. Sur le GF108, tout comme sur les GPU GF106 et GF104, NVIDIA a ajouté des unités d'exécutions à plus faible coût et a augmenté le ratio d'unités de *texturing* par rapport aux unités de calcul. Les multiprocesseurs sont maintenant composés de 48 unités de calcul et 8 unités de *texturing*. Cette stratégie a été employée pour augmenter le rendement en rendu 3D principalement pour les jeux vidéos. Ces GPUs visent le marché des cartes graphiques d'entrée et de milieu de gamme pour joueurs, elles doivent donc être proposées à des tarifs très agressifs. Plus en détail, un multiprocesseur de GF108 disposent de 2 doubles ordonnanceurs et 6 blocs d'exécution :

- trois unités SIMD 16-way (48 cœurs) capablent de traiter 32 opérations FMA FP32, 32 ADD INT32, 16 MUL INT32
- une unité SFU pouvant traiter 8 fonctions spéciales (cos, sqrt...) FP32 ou 16 interpolations
- une unité Load/Store 16-way 32 bits

Le GPU GF108 perd la possibilité d'effectuer des calculs en double précision à demi-vitesse, et peut envoyer 4 instructions par multiprocesseurs à chaque cycle contre 2 précédemment. En revanche, si chaque multiprocesseur du GF100 peut lancer 2 instructions par cycle, elles s'exécutent sur 2 groupes de données différents, il n'y a donc pas de dépendance entre les instructions et cela garantit un rendement optimale. Pour alimenter ses 16 unités supplémentaires, le GF108 peut quant à lui lancer 4 instructions par multiprocesseur, plus précisément, chacun des 2 ordonnanceurs peut lancer 2 instructions par groupe de 32 éléments mais celles-ci ne doivent pas dépendre l'une de l'autre. Dans le cas contraire, si toutes les instructions sont scalaires et dépendantes, le rendement tombera à 66%. Les deux double-ordonnanceurs se comporteront comme des ordonnanceurs et seules 32 unités de traitement pourront être exploitées [87].

Nous allons mesurer la puissance de calcul pratique du GPU de la carte CARMA pour voir si nous retrouvons cet effet. Pour cela, nous utilisons toujours la fonction *gemm* en simple précision complexe. La Figure 3.14 montre que la puissance de calcul atteinte par

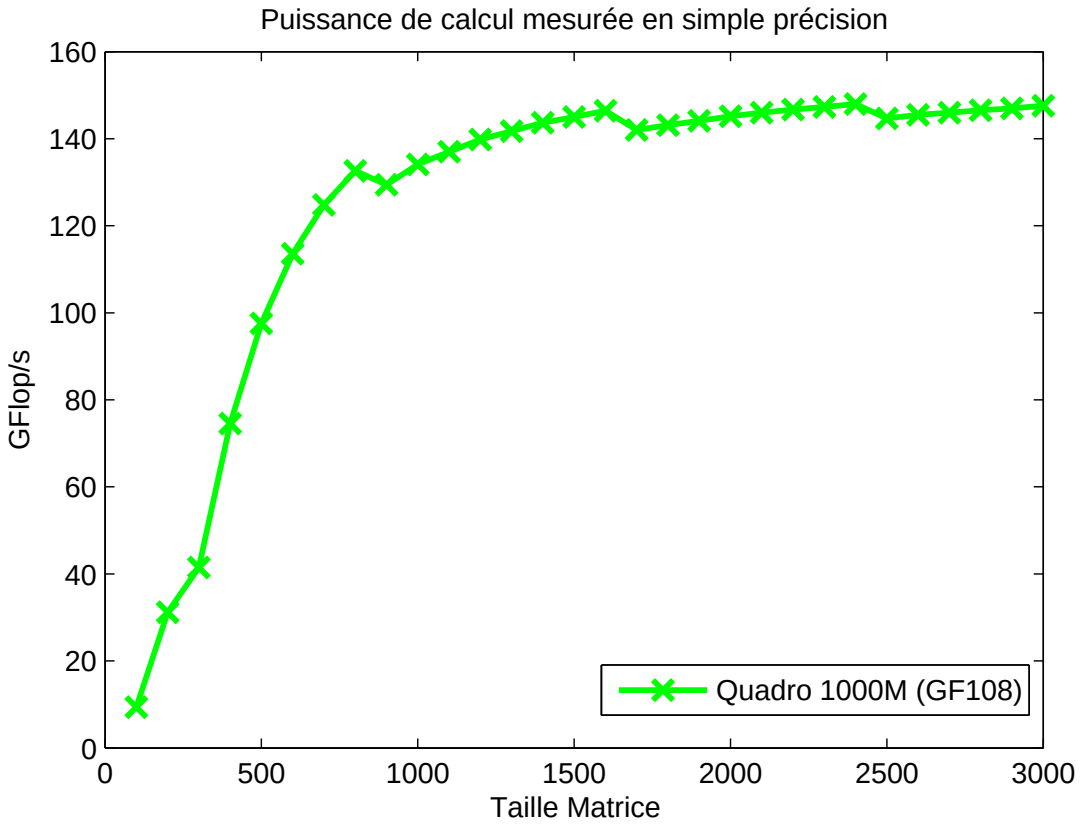


FIGURE 3.14 – Puissance de calcul mesurée du GPU de la carte CARMA.

le GPU GF108 de la carte CARMA tend vers environ 150 GFlop/s pour des grandes matrices. La puissance théorique de ce GPU étant de 268 GFlop/s, cela donne un rendement de 55%. Nous avions mesuré le GF100 de la carte Tesla C2050 à environ 800 GFlop/s (voir Figure 3.2) et un rendement de 78 %. Nous constatons donc que le rendement du GPU GF108 est égal comme prévu à presque 2/3 du rendement du GPU GF100 ($78*2/3 = 52$). Nous allons voir dans la Section suivante quel est l'impact de cette baisse de rendement sur les performances et l'efficacité énergétique de ce GPU dans le traitement STAP.

3.3.2 Performance et efficacité énergétique en traitement STAP

Nous testons ici les deux premiers étages du traitement STAP décrit précédemment. Ensemble, ces deux étages comprennent plus de 95% de la charge de calcul du traitement STAP. Pour ces essais, le processeur central est toujours un CPU Intel Xeon X5570, le GPU est toujours une carte NVIDIA Tesla C2050. Nous comparons les vitesses d'exécution du traitement s'exécutant sur ces deux processeurs avec les vitesses d'exécution du traitement sur le GPU de la carte CARMA.

Le tableau suivant présente les temps d'exécution de ces deux étages de traitement. Les calculs sont faits en simple précision.

Plateforme	Estimation	Inversion
CPU x86 Intel Xeon X5570	**72.67** ms	**54.40** ms
GPU NVIDIA Tesla C2050	**11.46** ms	**14.1** ms
NVIDIA CARMA	**62.41** ms	**84.1** ms

Les résultats montrent qu'un processeur généraliste classique (à instructions x86) et une carte CARMA ont des performances très proches sur ces deux étapes de traitement. Le GPU de la carte Tesla C2050 est par contre 3,8 à 6,3 fois plus rapide.

Nous avons aussi mesuré la consommation électrique de la carte CARMA et du système CPU-GPU (PC). Pour ce dernier, nous avons soustrait à la consommation mesurée les consommations des périphériques de refroidissements et de stockage. La consommation électrique du PC au repos est mesurée à $P = 146.5\,W$, alors qu'elle est de $P = 11.5\,W$ pour la carte CARMA. Lorsque le processeur central est utilisé pour l'un des étages de traitement STAP, la consommation du système monte à $P = 223\,W$, et à $P = 348\,W$ lorsque le GPU est utilisé. Sur la carte CARMA, lors du traitement STAP, la consommation totale de la carte est $P = 37\,W$. Nous avons ensuite calculé la performance par Watt pour les deux étages de traitement STAP. Nous divisons alors la charge de calcul des deux étages (voir Chapitre 1) par le temps d'exécution et par la consommation pour obtenir un résultat en GFlop/s/Watt. Le tableau suivant ainsi que les Figures 3.15 et 3.16 présentent les temps d'exécution et les efficacités des différentes plateformes PC en utilisant soit le CPU soit le GPU et carte CARMA avec son GPU.

Platform	Estimation	Inversion
Système x86 PC (CPU)	**0.24** GFlop/s/W	**0.17** GFlop/s/W
Système x86 PC (GPU)	**0.99** GFlop/s/W	**0.25** GFlop/s/W
Carte NVIDIA CARMA	**1.71** GFlop/s/W	**0.68** GFlop/s/W

Bien que n'étant pas la solution la plus rapide, la carte CARMA est presque deux fois plus efficace que la carte GPU haute performance fonctionnant sur PC. Ce système a lui-même une efficacité bien plus élevée si nous utilisons le GPU que si nous utilisons seulement le CPU. Notons que le gain en rapidité d'exécution apporté par les GPUs est plus élevé pour l'estimation des matrices de covariance que pour les inversions des matrices de covariance. Cela n'affecte pas les performances du CPU qui est conçu pour être rapide sur tout type de tâches.

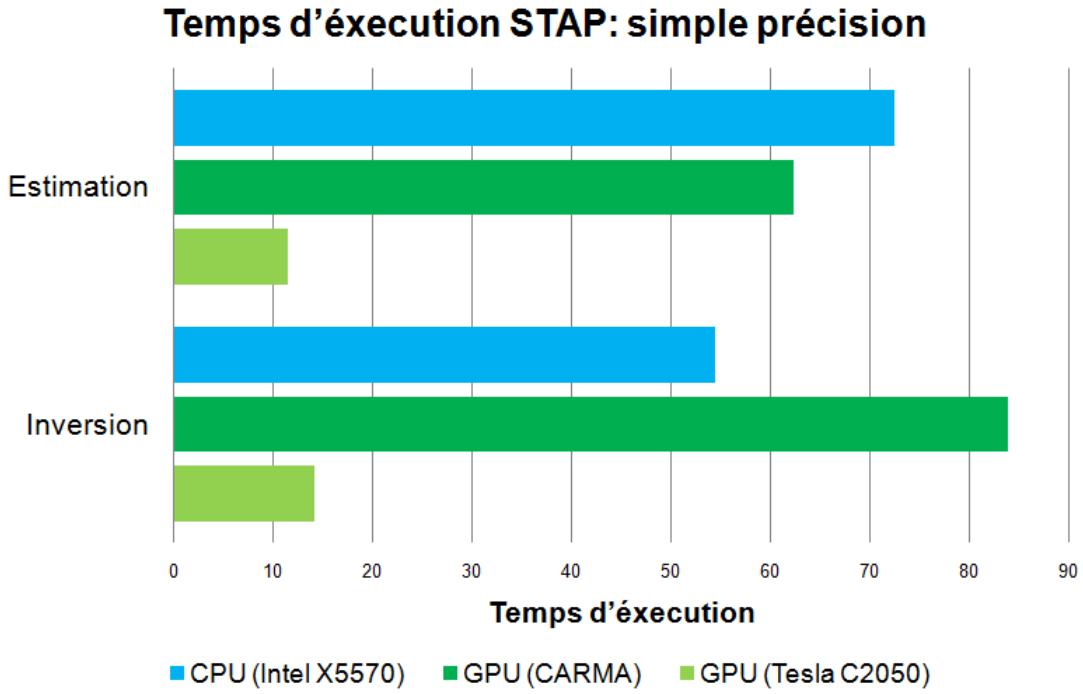


FIGURE 3.15 – Temps de calcul du système PC (CPU et GPU) et de la carte CARMA pour l'estimation et l'inversion

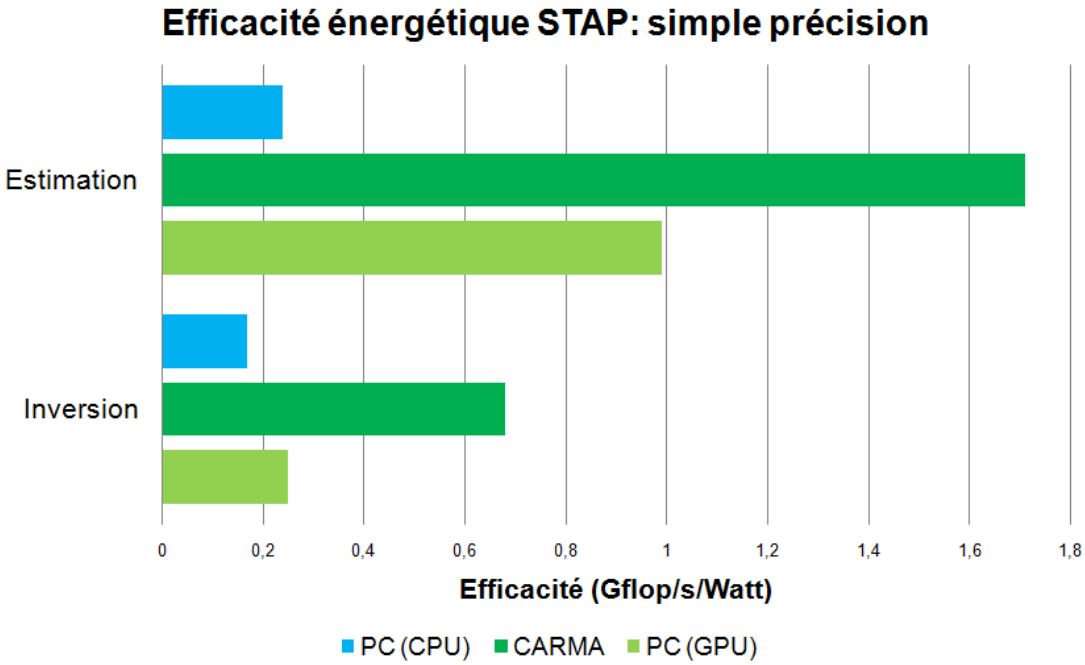


FIGURE 3.16 – Efficacité du système PC (CPU et GPU) et de la carte CARMA pour l'estimation et l'inversion

3.4 Conclusion

Dans ce chapitre, nous avons décrit les implémentations des différents algorithmes utilisés par un traitement STAP générique et mesuré leurs performances. La difficulté d'exploiter efficacement un GPU avec un grand nombre de petites matrices a été résolu en utilisant un modèle de programmation par flux. L'utilisation du GPU permet alors un gain significatif en performance par rapport à une solution traditionnelle pour un coût en temps de programmation réduit. Nous avons aussi montré qu'un facteur souvent cité comme limitant sur le GPU, le débit des données entre le processeur central et la carte GPU, ne pose aucun problème pour notre application STAP.

De plus, nous avons aussi mesuré les performances de certains de ces algorithmes sur une carte ayant une architecture originale composée d'un processeur ARM et d'un GPU. Avec la croissance accélérée des périphériques smartphones et tablettes, les processeurs à architecture ARM tendent à devenir de plus en plus efficaces et à équiper de plus en plus de systèmes portables. Les résultats sur nos algorithmes de traitement STAP pointent le fait que, couplés à des GPUs, ces processeurs sont une alternative très intéressante, en terme d'efficacité énergétique, aux systèmes basés sur des CPUs x86 traditionnels.

Conclusion générale

Les travaux menés au cours de cette thèse ont permis de montrer que les méthodes à données primaires seules sont particulièrement adaptées au cas d'environnements fortement hétérogènes dans lesquels la représentativité des données d'apprentissage est prise en défaut. Nous avons proposé une extension d'une de ces méthodes, le détecteur MLED, que nous avons appelé "Stop-Band APES", et qui permet de réduire considérablement la charge de calcul du détecteur MLED, en supprimant ses propriétés intrinsèques d'hyper-résolution. Ce nouveau détecteur a démontré son efficacité sur des données réalistes.

Nous avons ensuite proposé deux versions plus robuste de cet algorithme basées sur les méthodes de sous-espaces, ce qui permet au traitement STAP de requérir moins de données pour l'estimation de la matrice de covariance. Dans les situations hétérogènes, les méthodes traditionnelles manquent de données d'entrainement en nombre suffisant, impactant ainsi négativement les performances du traitement STAP. L'aspect négatif des méthodes de sous-espaces est la complexité calculatoire, qui est habituellement bien plus élevée que celle des méthodes classiques. Ce point est important pour une utilisation pratique. Nous avons donc proposé une nouvelle méthode qui possède à la fois les performances des méthodes de sous-espaces et un coût calculatoire réduit.

Après avoir mis en évidence la légère diminution de la réjection du fouillis lorsque nous utilisons des méthodes basées sur l'algorithme APES, nous avons proposé deux méthodes basées sur le traitement déterministe pour corriger cet effet. La première, qui se sert de la relation théorique angle-Doppler, est particulièrement bien adaptée pour la détection de cibles en mode air-sol. Les résultats sur des données réelles montrent qu'elle surpasse aussi bien le traitement STAP générique que les méthodes basées sur APES. La seconde méthode, dont le but est de supprimer la composante continue des interférences, est quant à elle plus adaptée au mode air-air. Les résultats sur des données réalistes montrent qu'elle parvient à réduire sensiblement le nombre de fausses alarmes par rapport aux méthodes classiques et aux méthodes précédemment proposées.

Enfin, nous avons proposé une nouvelle approche pour la sélection des données d'entrainement, basée sur une recherche des matrices optimales pour calculer les poids du filtre STAP. Nous avons aussi établi le lien entre cette méthode et la géométrie Riemannienne. Cette nouvelle approche a montré son efficacité sur des données synthétiques réalistes ainsi que sur des données réelles.

Dans une deuxième partie de la thèse, nous nous sommes intéressés à l'implémentation matérielle d'un traitement STAP générique. Nous avons mis en évidence les limitations liées à l'implémentation efficace d'un traitement STAP. L'architecture des GPUs leur permet d'être efficaces avec une implémentation simple sur des traitements parallèles travaillant sur de gros volumes de données, ce qui n'est pas le cas des traitements STAP.

Nous avons développés plusieurs algorithmes, basés sur une parallélisation à deux niveaux, des différentes étapes d'un traitement STAP qui permettent d'atteindre sur GPU des performances bien plus élevées qu'avec un processeur généraliste CPU. Nous avons aussi mesuré les performances de ces implémentation sur une carte ayant une architecture originale ARM+GPU. Les résultats par rapport à la consommation mesurée de cette plateforme rendent l'utilisation de ce couple de processeurs très intéressant pour des systèmes embarqués.

Les résultats des différentes méthodes développées pour surmonter les problèmes liées aux environnements hétérogènes, permettent d'envisager une utilisation de traitement STAP en situation pratique. L'impact de ces traitements sur la localisation des cibles détectées doit cependant être étudié. Certains aspects, comme la stratégie de sélection des données d'entrainement peuvent être approfondis. Pour ce dernier point, la géométrie de l'information offre des perspectives intéressantes.

L'arrivée de nouvelles générations de GPUs, prenant en compte de plus en plus le calcul générique lors de leur conception, offre des possibilités encore accrues pour des implémentations efficaces de traitements STAP. En particulier, les futures plateformes ARM-GPU dont le prototype CARMA est un précurseur, représentent des solutions très prometteuses pour des traitements lourds et parallèles sur des systèmes embarqués.

A

Contexte et état de l'art

A.1 Généralités

A.1.1 Équation du radar

L'équation du radar ou bilan radar est le rapport entre la puissance émise par l'antenne du système et la puissance reçue par les capteurs.

$$\frac{P_r}{P_e} = \frac{G_r G_e \lambda^2 \sigma}{(4\pi)^3 D^4} \tag{A.1}$$

où G_r et G_e sont les gains d'antenne en réception et en émission, λ est la longueur d'onde et D est la distance radar-cible. La surface équivalente radar (SER) σ, est caractéristique de la réflectivité de la cible. Plus la SER est importante, plus la cible réfléchit l'onde électromagnétique.

A.1.2 Bruit thermique

Le bruit dit thermique, inhérent à tout matériel électronique, vient de l'agitation des électrons dans les différents constituants du système radar. Même si l'antenne est isolée de l'environnement et ne reçoit aucun signal, le bruit thermique sera tout de même présent. Le bruit thermique est considéré comme la limite des traitements possibles, il n'est pas possible de le supprimer. Ce bruit est considéré comme blanc, et s'écrit

$$b = kT_0W$$

avec k la constante de Boltzmann ($k = 1.3810^{-23}\, J/K$), T_0 la température (en degrés Kelvin K) et W la bande de fréquence considérée (Hz). Nous supposons que ce bruit est décorrélé entre capteurs et de puissance σ^2. La vraie matrice de covariance du bruit thermique est définie par

$$\mathbf{\Gamma_{th}} = \sigma^2 \mathbf{I}_{NM} \tag{A.2}$$

Lorsque nous n'avons pas accès à la vraie matrice de covariance du bruit thermique $\mathbf{\Gamma_{th}}$, nous en calculons une estimée $\mathbf{R_{th}}$ sur un domaine qui doit être préservé de la présence de cibles ou d'interférences (fouillis, brouilleurs). Le vecteur de données $\mathbf{x}$ est alors composé uniquement de bruit thermique ($\mathbf{x} = \mathbf{n}$) que l'on peut considérer comme un vecteur aléatoire gaussien centré ayant pour matrice de covariance $\mathbf{\Gamma_{th}}$ et est noté $\mathbf{n} \sim \mathcal{CN}(0, \mathbf{\Gamma_{th}})$.

A.1.3 Le détecteur GLRT de Kelly

Le détecteur *Generalized Likelihood Ratio Test* (GLRT) de Kelly utilise à la fois les données primaires et les données secondaires. Il est défini par

$$\frac{|\mathbf{s_s^H R^{-1} x}|^2}{(\mathbf{s_s^H R^{-1} s_s})(1 + \mathbf{x^H R^{-1} x})} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta, \quad \eta \in [0; 1] \tag{A.3}$$

Ce détecteur possède la propriété CFAR.

A.2 Antennes radar

Nous considérons dans la thèse deux types d'antennes : les antennes à visée latérale et les antennes à visée frontale (de pointe avant).

A.2.1 Antennes de pointe avant

Les traitements STAP sont envisageable avec l'arrivée d'antennes à balayage électronique actives (*Active Electronically Scanned Array*, AESA) parce qu'elle comportent plusieurs sous-réseaux de réception. Ces antennes sont composées d'un très grand nombre de modules émetteurs-récepteurs, regroupés en sous-réseaux. En jouant sur le déphasage des signaux émis par chaque module, le radar dirige le ou les faisceaux d'émission. De la même manière une première formation de faisceaux est effectuée pour chaque sous-réseau, et il est alors possible de former des faisceaux par le calcul à l'intérieur des lobes de sous-réseaux. Ces antennes sont généralement des réseaux surfaciques de forme circulaire.

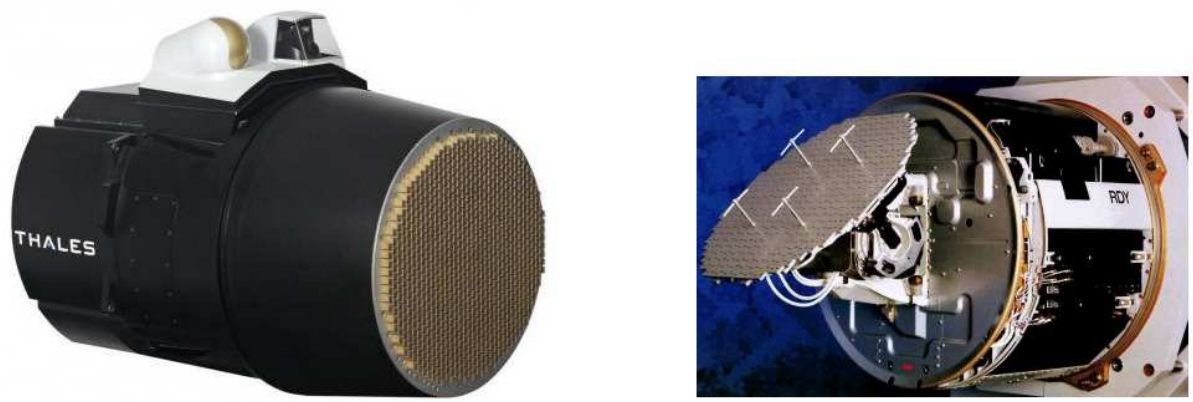


FIGURE A.1 – Illustration d'un radar à balayage électronique Thales RBE2-AESA (à gauche) et d'un radar à balayage mécanique Thales RDY (à droite). Source : Thales.

A.2.2 Antennes à visée latérale

Les antennes radar à visée latérale sont des antennes montées sur un avion ou un satellite, qui sont dirigées vers le sol dans une direction perpendiculaire au déplacement du porteur. Elles sont constituées de capteurs, ou de réseaux de capteurs, linéairement et uniformément répartis sur un axe qui est proche de la direction du porteur (il est égal à la direction du porteur si l'angle de dérive est nul). Ainsi il n'est pas possible de contrôler l'angle d'élévation par le calcul.

Ces types d'antennes sont particulièrement bien adaptées aux traitement radar à ouverture synthétique (SAR) où le déplacement du porteur est utilisé pour émuler une antenne de très grand diamètre et obtenir une image radar haute résolution après traitement [88].

A.3 Données radar utilisées

A.3.1 ONERA

Les données **ONERA-AS**, **ONERA-ALU** et **ONERA-AA** sont des données synthétiques réalistes. Elles prennent en compte la découpe spécifique en sous-réseau ainsi que les erreurs de gain/phases des éléments rayonnants. Du fait que ces données proviennent d'un simulateur, nous avons accès aux vraies matrices de covariance $\mathbf{\Gamma}$ et $\mathbf{\Gamma_{th}}$ ainsi qu'à la position des cibles.

ONERA-AS

Les données **ONERA-AS** correspondent à un radar de pointe avant en mode air-sol. L'antenne simulée est une antenne de type AMSAR [89], illustrée sur la Figure A.5. Les paramètres de la configuration utilisée sont détaillés dans le tableau suivant

Porteur	
vitesse	$V_a = 180$ m/s
altitude	$h = 3000$ m
Forme d'onde	
type	*chirp*
fréquence porteuse	$f_0 = 10$ GHz
largeur bande	$B = 2$ MHz
résolution distance	$\delta_r = c/2B = 75$ m
PRF	$f_{PRF} = 2$ kHz
Nombre d'impulsions	$M_p = 64$
Antenne	
Type	Circulaire
Nombre de voies	$N = 8$
Ouverture	$\approx 4^o$
Pointage	
élévation	$\phi - 3^o$
azimut	$\theta = 30^o$

Tableau 1 : Paramètres de données **ONERA-AS**

La voie somme est obtenue simplement par sommation sur la dimension spatiale (ce qui correspond à un pointage dans la direction visée). La Figure A.2 montre une image distance-vitesse de la voie somme pour la configuration du Tableau 1. Elle présente notamment un fouillis en provenance de la deuxième ambiguïté distance (>75km) et des lobes secondaires pour les distance allant de 10 à 37 km. Ensuite de 37 à 45 km, le fouillis est gaussien de niveau σ_0. Puis, entre 45 km et 52 km, se trouve un fouillis piqué hétérogène. Nous retrouvons un fouillis gaussien homogène dans la tranche 52-60 km puis un fouillis gaussien de niveau plus faible $\sigma_0/10$ entre 60 et 68 km. Enfin, nous retrouvons le fouillis gaussien de niveau σ_0 de 68 à 75 km. La Figure A.2 montre aussi les positions exactes

des cibles (ronds blancs) et des échos ponctuels forts et fixes (triangles blancs). Un convoi composé de 10 cibles mobiles de même vitesse est présent à la distance de 60 km.

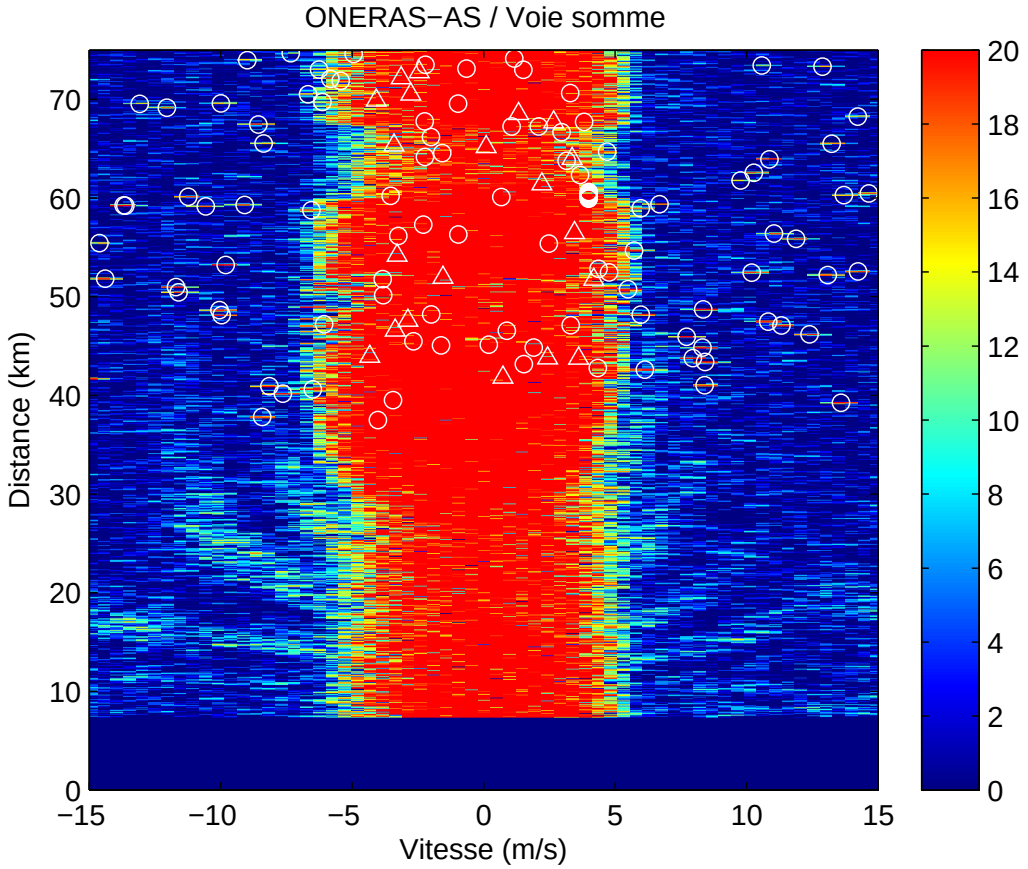


FIGURE A.2 – Image distance-vitesse de la voie somme (données **ONERAS-AS**)

ONERA-ALU

Les données **ONERA-ALU** représentent un configuration d'antenne linéaire uniforme à visée latérale et un mode radar air-sol. Elles sont issues du même simulateur que les données **ONERA-AS** mais la configuration est différente. L'antenne est cette fois à visée latérale, la résolution en distance est supérieure. La densité de cible est donc beaucoup plus importante que pour les données **ONERA-AS**. La configuration utilisée est présentée sur le Tableau 2.

La Figure A.3 montre la voie somme pour les données **ONERA-ALU**. Nous voyons que la densité de cibles est très élevée (ronds blancs). Contrairement aux données **ONERA-AS**, le fouillis est gaussien et de même niveau sur toute la dimension distance. En revanche, la détection de cibles de faible vitesse est compliquée par le présence de lobes secondaires assez forts (l'antenne n'est pas pondérée). Un convoi composé de 20 cibles mobiles de même vitesse est présent à la distance de 60 km.

Porteur	
vitesse	$V_a = 100$ m/s
altitude	$h = 4500$ m
Forme d'onde	
type	*chirp*
fréquence porteuse	$f_0 = 10$ GHz
largeur bande	$B = 10$ MHz
résolution distance	$\delta_r = c/2B = 15$ m
PRF	$f_{PRF} = 2$ kHz
Nombre d'impulsions	$M_p = 64$
Antenne	
Type	Linéaire uniforme (latérale)
Nombre de voies	$N = 4$
Ouverture (azimut)	$\approx 1,7^o$
Pointage	
azimut	$\theta = 0^o$
élévation	$\phi = -3,7^o$

Tableau 2 : Paramètres de données **ONERA-ALU**

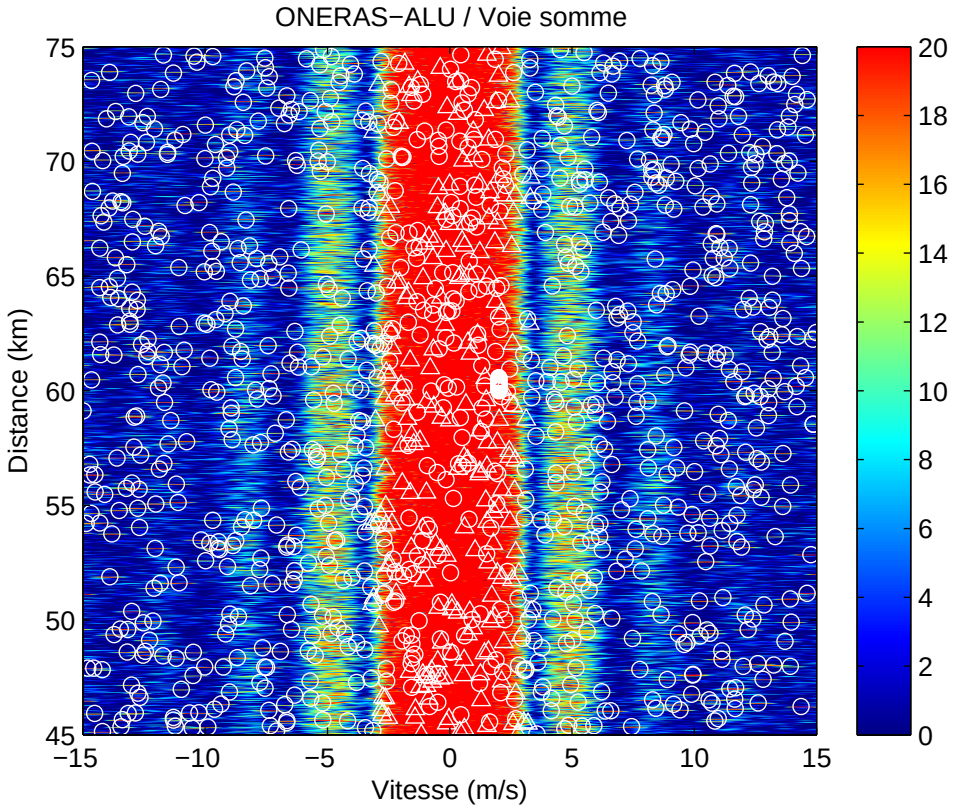


FIGURE A.3 – Image distance-vitesse de la voie somme (données **ONERA-ALU**)

ONERAS-AA

Enfin, toujours obtenues avec le même simulateur, les données **ONERAS-AA** émulent un scénario de mode air-air en configuration radar pointe avant. L'antenne simulée est encore l'antenne AMSAR, dont la découpe est illustrée sur le Figure A.5. Les paramètres de la configuration utilisée sont détaillés dans le tableau suivant

Porteur	
vitesse	$V_a = 300$ m/s
altitude	$h = 1500$ m
Forme d'onde	
type	*chirp*
fréquence porteuse	$f_0 = 10$ GHz
largeur bande	$B = 2$ MHz
résolution distance	$\delta_r = c/2B = 75$ m
PRF	$f_{PRF} = 20$ kHz
Nombre d'impulsions	$M_p = 128$
Antenne	
Type	Circulaire
Nombre de voies	$N = 8$
Ouverture	$\approx 4^o$
Pointage	
élévation	$\phi = 0^o$
azimut	$\theta = 30^o$

Tableau 3 : Paramètres de données **ONERA-AA**

La Figure A.4 montre une image distance-vitesse de la voie somme obtenue sur ces données. Le mode air-air est caractérisé par la présence de peu de cibles, seulement 5 cibles sont présentes dans la scène aux distances suivantes :

Cibles	Vitesse (m/s)	Distance (km)
1	50.03	58.425
2	100.22	55.425
3	115.026	57.00
4	185.0265	57.30
5	216.0296	59.475

Le fouillis arrivant par le lobe principal est homogène en distance (raie centrée à 260 m/s). Le fouillis arrivant par les lobes secondaires est en revanche fortement hétérogène et rend la zone située entre 54 et 55,5 km quasiment aveugle. Ce fouillis est aussi présent aux vitesse voisines à celles du lobe principal dans les cases distances suivantes.

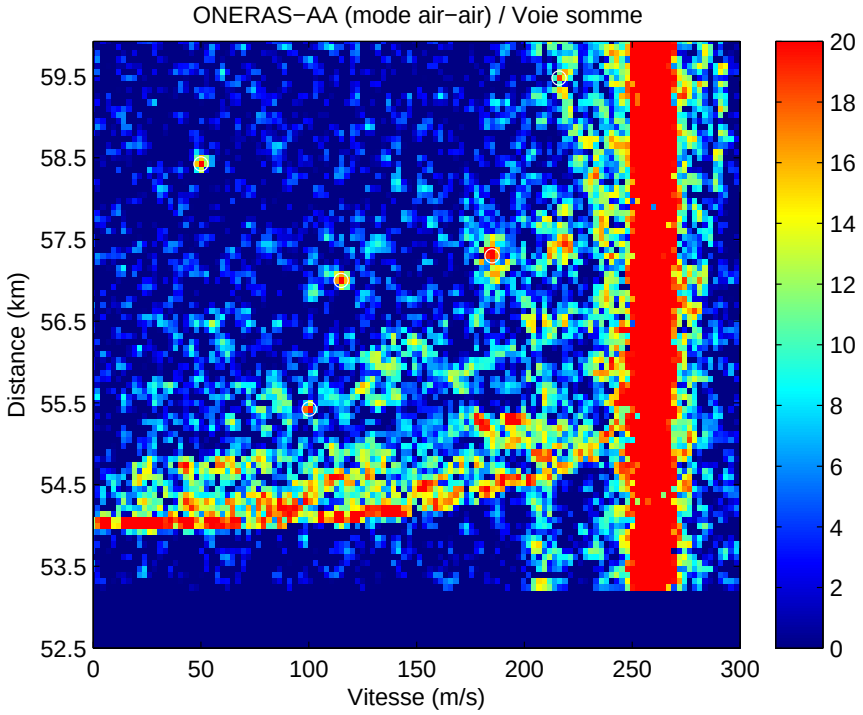


FIGURE A.4 – Image distance-vitesse de la voie somme (données **ONERA-AA**)

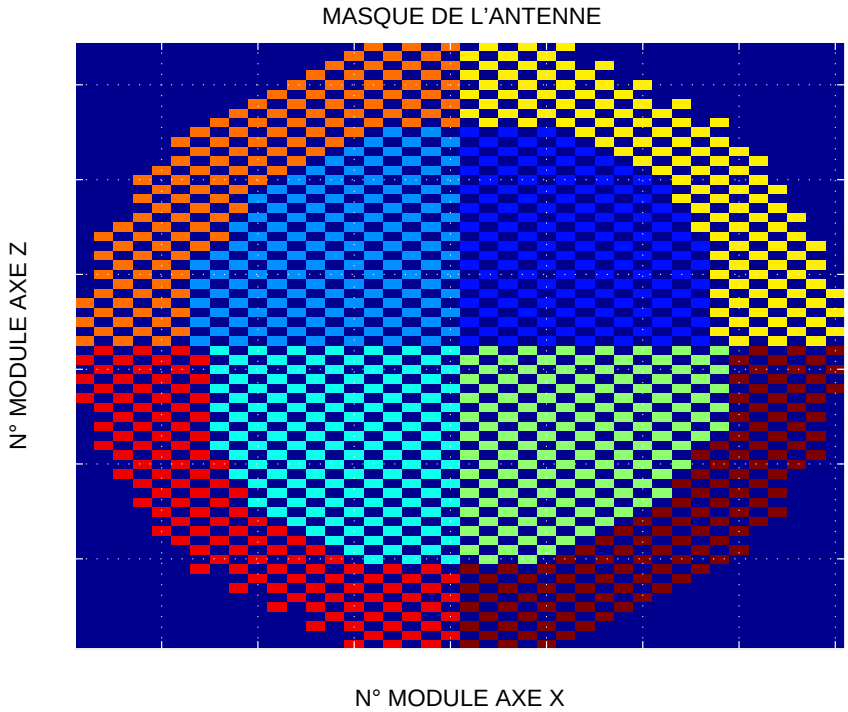


FIGURE A.5 – Découpe en sous-réseaux de l'antenne simulée de type AMSAR

A.3.2 DGA/MI

Les données **DGA/MI** (anciennement CELAR) sont des données hybrides correspondant à une configuration d'antenne linéaire uniforme à visée latérale en mode air-sol. Les données que nous utiliserons correspondent à l'essai 10 du jeu de données DGA/MI décrit dans [90]. Ces données ont été obtenues avec le simulateur SIROS [91]. Si les cibles sont synthétiques, le fouillis est quant à lui réel et obtenu à partir de données collectées en mode SAR à très haute résolution (programme THR RAMSES) dont on a dégradé les résolutions radiale et transverse. Les paramètres de configuration pour ces données sont les suivants :

Porteur	
vitesse	$V_a = 100$ m/s
altitude	$h = 2600$ m
Forme d'onde	
type	*chirp*
fréquence porteuse	$f_0 = 10$ GHz
largeur bande	$B = 4,43$ MHz
résolution distance	$\delta_r = c/2B = 30$ m
PRF	$f_{PRF} = 1$ kHz
Nombre d'impulsions	$M_p = 64$
Antenne	
Type	Linéaire uniforme
Nombre de voies	$N = 4$
Ouverture	$\approx 1,9^o$
Pointage	
élévation	$\phi - 5^o$
azimut	$\theta = 0^o$

Tableau 4 : Paramètres de données **DGA/MI**

La Figure A.6 montre la sortie de la voie somme pour les données **DGA/MI**. Comme nous pouvons le voir, le scénario comporte 3 cibles qui sont toutes exo-fouillis, c'est à dire qu'elle sont détectables à l'aide d'un simple traitement Doppler classique. Les positions des 3 cibles sont les suivantes :

Cibles	Vitesse (m/s)	Case distance
1	4	216
2	4	256
3	-4	296

Ces données permettent de vérifier que le traitement STAP supprime correctement le fouillis, tout en conservant les cibles qui étaient déjà détectables avec un traitement Doppler.

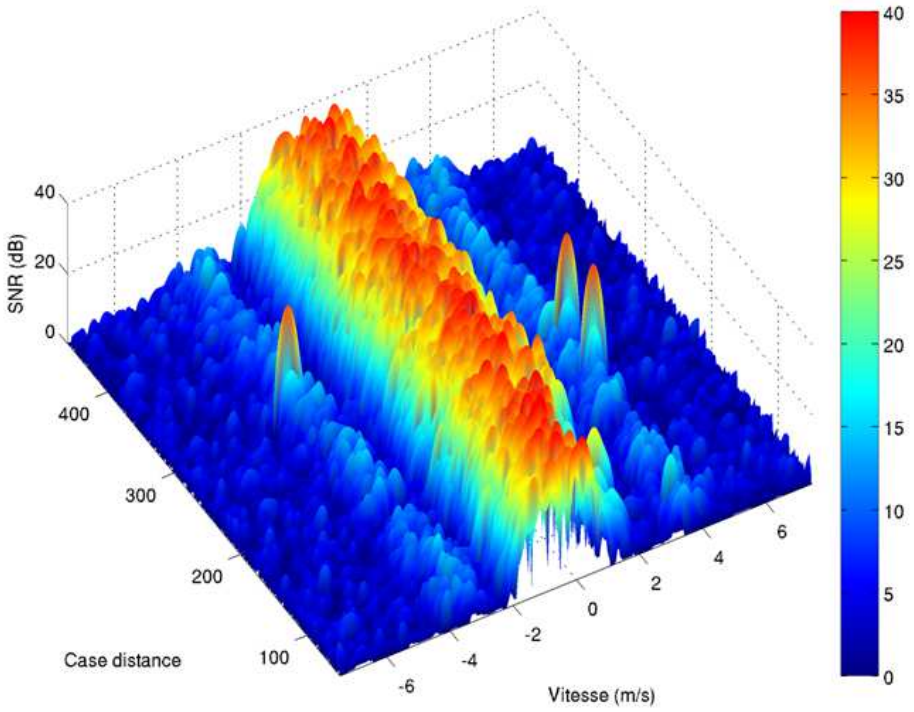


FIGURE A.6 – Image distance-vitesse de la voie somme (données **DGA/MI**)

A.3.3 SimALU

Les données **SimALU** sont des données que nous construisons de manière totalement théorique et sans ajouter d'imperfection. Nous simulons une antenne linéaire uniforme à visée latérale. Le fouillis est gaussien, homogène en distance et suit la loi de l'Equation (1.1) du Chapitre 1.2.1, Partie 1. Nous pouvons choisir d'ajouter une cible à n'importe quelle position souhaitée. Les données étant simulées, nous avons accès à la vraie matrice de covariance. Les paramètres peuvent dépendre des simulations et sont alors précisés, mais certains sont communs à toutes les simulations :

Porteur	
vitesse altitude	$V_a = 100$ m/s $h = 9000$ m
Forme d'onde	
fréquence porteuse	$f_0 = 10$ GHz
Antenne	
Type	Linéaire uniforme
Pointage	
azimut	$\theta = 0^o$

Tableau 5 : Paramètres communs aux données **SimALU**

A.3.4 SAREX

Les données **SAREX** sont des données réelles acquises par l'ONERA à l'aide du radar RAMSES [92]. Ces données comprennent 39 rafales, nous avons choisi la rafale 8 dont la voie somme est affichée sur la Figure A.7. Les paramètres de la configuration radar sont les suivants :

Porteur	
vitesse	$V_a = 85$ m/s
Forme d'onde	
type	*chirp*
fréquence porteuse	$f_0 = 10$ GHz
largeur bande	$B = 15$ MHz
résolution distance	$\delta_r = c/2B = 3,75$ m
PRF	$f_{PRF} = 3,125$ kHz
Nombre d'impulsions	$M_p = 64$
Antenne	
Type	Linéaire uniforme
Nombre de voies	$N = 4$
Ouverture	$\approx 2,4^o$
Pointage	
azimut	$\theta = 0^o$

Tableau 6 : Paramètres de données **SAREX**

L'antenne 4 voies utilisée est du type Σ, Δ, Δ', Δ''. De façon simplifiée, si $SR1$, $SR2$, $SR3$ et $SR4$ sont les 4 sous-réseaux adjacents de longueur totale de 70 cm, les voies correspondent à :

$$\begin{aligned}\Sigma &= SR1 + SR2 + SR3 + SR4\\ \Delta &= SR1 + SR2 - SR3 - SR4\\ \Delta' &= -SR1 + SR2 + SR3 - SR4\\ \Delta'' &= -SR1 + SR2 - SR3 + SR4\end{aligned}$$

Nous ne possédons pas de vérité terrain mais à l'aide des rafales qui précèdent et suivent la rafale 8, ainsi qu'aux traitement STAP, nous pouvons faire l'hypothèse que 3 cibles se trouvent sur cette rafale :

Cibles	Vitesse (m/s)	Case distance
1	5	138
2	6	149
3	3	214

La Figure A.7 montre la voie somme (Σ) pour la rafale 8, alors que la Figure A.8 montre la voie différence (Δ) pour la même rafale.

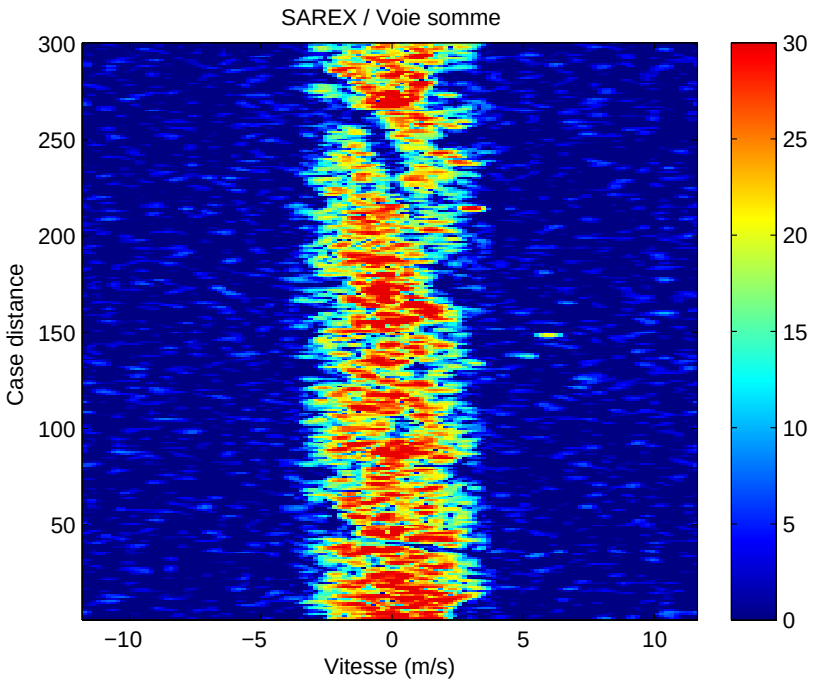


FIGURE A.7 – Image distance-vitesse de la voie somme (données **SAREX**)

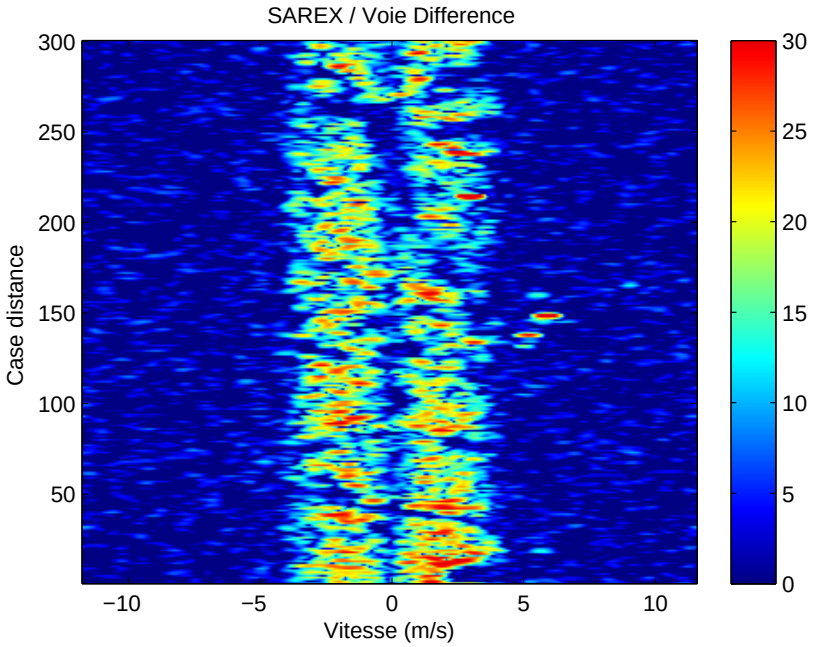


FIGURE A.8 – Image distance-vitesse de la voie différence (données **SAREX**)

B

Traitement STAP

B.1 Démonstration détecteur MLED

La problématique est de trouver le filtre $\mathbf{w}$ tel que :

$$\min_{\mathbf{w},\alpha} \quad \left(\mathbf{w^H X} - \alpha \mathbf{s_t^T}\right)\left(\mathbf{w^H X} - \alpha \mathbf{s_t^T}\right)^H \tag{B.1}$$
$$sous\, contrainte\ :\ \mathbf{w^H C} = \mathbf{f}$$

où $\mathbf{X}$ est la matrice de données de taille $NM \times K_t$, s_t est le *steering* vecteur temporel (Doppler) de taille $M_p - M + 1 = K_t$ et $w^H C = f$ est la contrainte.

$$\begin{aligned}
& \left(\mathbf{w^H X} - \alpha \mathbf{s_t^T}\right)\left(\mathbf{w^H X} - \alpha \mathbf{s_t^T}\right)^H \\
= \ & \mathbf{w^H X X^H w} + \alpha \mathbf{s_t^T s_t^*} \alpha^H - \alpha \mathbf{s_t^T X^H w} - \mathbf{w^H X s_t^*} \alpha^H \\
Comme \quad & \mathbf{s_t^T s_t^*} = K_t \\
& \left(\alpha\sqrt{K_t} - \frac{1}{\sqrt{K_t}} \mathbf{w^H X s_t^*}\right)\left(\alpha\sqrt{K_t} - \frac{1}{\sqrt{K_t}} \mathbf{w^H X s_t^*}\right)^H \\
= \ & \alpha K_t \alpha^H + \frac{1}{K_t} \mathbf{w^H X s_t^* s_t^T X^H w} - \alpha \mathbf{s_t^T X^H w} - \mathbf{w^H X s_t^*} \alpha^H \\
Soit\ : \quad & \left(\left(\mathbf{w^H X} - \alpha \mathbf{s_t^T}\right)\left(\mathbf{w^H X} - \alpha \mathbf{s_t^T}\right)^H\right. \\
= \ & \left(\alpha\sqrt{K_t} - \frac{1}{\sqrt{K_t}} \mathbf{w^H X s_t^*}\right) \\
& \left(\alpha\sqrt{K_t} - \frac{1}{\sqrt{K_t}} \mathbf{w^H X s_t^*}\right)^H + \mathbf{w^H X X^H w} - \frac{1}{K_t} \mathbf{w^H X s_t^* s_t^T X^H w}
\end{aligned}$$

La contrainte ne dépendant pas de α, minimiser le critère selon α revient à minimiser $\left(\alpha\sqrt{K_t} - \frac{1}{\sqrt{K_t}} \mathbf{w^H X s_t^*}\right)\left(\alpha\sqrt{K_t} - \frac{1}{\sqrt{K_t}} \mathbf{w^H X s_t^*}\right)^H$, ce qui donne :

$$\alpha = \frac{\mathbf{w^H X s_t^*}}{K_t}$$

En ré-injectant cela dans l'équation (B.1), nous avons

$$\min_{\mathbf{w}} \quad \mathbf{w^H}\left(\mathbf{X} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{\mathbf{s_t^T s_t^*}}\right)\left(\mathbf{X} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{\mathbf{s_t^T s_t^*}}\right)^H \mathbf{w}$$
$$\text{tel que}\ :\ \mathbf{w^H C} = \mathbf{f}$$

Nous revenons à une minimisation sous contrainte et si nous notons $\mathbf{Q} = \left(\mathbf{X} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{\mathbf{s_t^T s_t^*}}\right)\left(\mathbf{X} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{\mathbf{s_t^T s_t^*}}\right)^H$, la solution s'écrit

$$\mathbf{w^H} = \mathbf{f^H}(\mathbf{C^H Q^{-1} C})^{-1}\mathbf{C^H Q^{-1}} \tag{B.2}$$

Avec la contrainte de gain unité dans la direction $w^H s_s = 1$, la solution générale B.2 s'écrit

$$\mathbf{w} = \frac{\mathbf{Q^{-1}s_s}}{\mathbf{s_s^H Q^{-1} s_s}} \tag{B.3}$$

Posons $g = \frac{1}{K_t} X s_t^*$, nous pouvons ré-écrire $\mathbf{Q}$

$$\begin{aligned}
\mathbf{Q} &= \left(\mathbf{X} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\right)\left(\mathbf{X} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\right)^H \\
\Leftrightarrow \mathbf{Q} &= \mathbf{X X^H} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\mathbf{X^H} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\mathbf{X^H} + \mathbf{X}\frac{\mathbf{s_t^* s_t^T s_t^* s_t^T}}{K_t^2}\mathbf{X^H} \\
\Leftrightarrow \mathbf{Q} &= \mathbf{X X^H} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\mathbf{X^H} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\mathbf{X^H} + \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\mathbf{X^H} \\
\Leftrightarrow \mathbf{Q} &= \mathbf{X X^H} - \mathbf{X}\frac{\mathbf{s_t^* s_t^T}}{K_t}\mathbf{X^H} \\
\Leftrightarrow \mathbf{Q} &= \mathbf{X X^H} - K_t \mathbf{g g^H}
\end{aligned}$$

Nous voyons que nous pouvons prendre n'importe quel multiple de $\mathbf{Q}$ sans changer les équations. Nous choisissons de prendre $\mathbf{Q} = \frac{1}{K_t}\mathbf{X X^H} - \mathbf{g g^H}$, et nous pouvons estimer la puissance de bruit par l'APR $\mathbf{w^H Q w}$.

$$\begin{aligned}
\mathbf{w^H Q w} &= \frac{\mathbf{s_s^H Q^{-1} Q Q^{-1} s_s}}{\mathbf{s_s^H Q^{-1} s_s s_s^H Q^{-1} s_s}} \\
&= \frac{1}{\mathbf{s_s^H Q^{-1} s_s}}
\end{aligned}$$

Finalement, le traitement se résume à filtrer $\mathbf{X}$

$$\mathbf{Y} = \frac{\mathbf{s_s^H Q^{-1} X}}{\mathbf{s_s^H Q^{-1} s_s}}$$

et à effectuer une corrélation de $\mathbf{Y}$ avec la réplique attendue après filtrage $Y s_t^*$, ce qui est proportionnel à g, et revient à une transformée de Fourier des données filtrées. Le test de détection s'écrit donc

$$\text{Détecteur MLED} : \frac{|\mathbf{s_s^H Q^{-1} g}|^2}{(\mathbf{s_s^H Q^{-1} s_s})} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta$$

B.2 Lemme d'inversion matricielle

Nous utiliserons également un autre résultat très utile pour inverser récursivement une matrice :

Lemme 1. *Soit* $\mathbf{A}$ *et* $\mathbf{C}$ *deux matrices inversibles et* $\mathbf{B}$ *et* $\mathbf{D}$ *deux matrices telles que*

$$\mathbf{X} = \mathbf{A} + \mathbf{BCD}$$

Il est alors possible de calculer la matrice $\mathbf{X}^{-1}$*, inverse de* $\mathbf{X}$ *de la manière suivante :*

$$\mathbf{X}^{-1} = (\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}[\mathbf{C}^{-1} + \mathbf{DA}^{-1}\mathbf{B}]^{-1}\mathbf{DA}^{-1}$$

Cette formule est également connue sous le nom de *formule de Sherman-Morrison-Woodbury.*

B.3 Réponse spectrale du projecteur APES

Intéressons nous à la réponse spectrale du filtre $\mathbf{X}\left(\mathbf{I} - \frac{\mathbf{s_t^* s_t^T}}{\mathbf{s_t^T s_t^*}}\right)$. Soit x une sinusoïde à la fréquence f_1 sur K_t points. Ce projecteur cherche à retirer la fréquence f_0 des données.

$$\begin{aligned}
x(k) &= e^{2\pi i k f_1} \\
s_t(k) &= e^{2\pi i k f_0} \\
y(k) &= x(k) - \frac{1}{K_t}\sum_{l=0}^{N-1} x(l)s_t(l)^* s_t(k) \\
&= x(k) - \frac{1}{K_t} e^{2\pi i k f_0} \sum_{l=0}^{N-1} \left(e^{2\pi i l f_1} e^{-2\pi i l f_0}\right) \\
\Leftrightarrow Y(f) &= X(f) - \frac{1}{K_t}\sum_{j=0}^{N-1}\left(e^{2\pi i j f_0} e^{-2\pi i j f} \sum_{l=0}^{N-1}\left(e^{2\pi i l f_1} e^{-2\pi i l f_0}\right)\right) \\
&= X(f) - \frac{1}{K_t}\sum_{j=0}^{N-1}\left(e^{2\pi i j f_0} e^{-2\pi i j f}\right) \sum_{l=0}^{N-1}\left(e^{2\pi i l f_1} e^{-2\pi i l f_0}\right) \\
&= X(f) - \frac{1}{K_t}\sum_{j=0}^{N-1}\left(e^{2\pi i j (f_0 - f)}\right) \sum_{l=0}^{N-1}\left(e^{2\pi i l (f_1 - f_0)}\right) \\
&= X(f) - \frac{1}{K_t}\left(\frac{1 - e^{2\pi i K_t (f_0 - f)}}{1 - e^{2\pi i (f_0 - f)}}\right)\left(\frac{1 - e^{2\pi i K_t (f_1 - f_0)}}{1 - e^{2\pi i (f_1 - f_0)}}\right) \\
&= X(f) - \frac{1}{K_t}\left(\frac{e^{\pi i K_t (f_0 - f)} sin(\pi K_t (f_0 - f))}{e^{\pi i (f_0 - f)} sin(\pi (f_0 - f))}\right)\left(\frac{e^{\pi i K_t (f_1 - f_0)} sin(\pi K_t (f_1 - f_0))}{e^{\pi i (f_1 - f_0)} sin(\pi (f_1 - f_0))}\right) \\
&= X(f) - \frac{1}{K_t} e^{\pi i (K_t - 1)(f - f_1)} \frac{sin(K_t \pi (f - f_0))}{sin(\pi (f - f_0))} \frac{sin(K_t \pi (f_1 - f_0))}{sin(\pi (f_1 - f_0))}
\end{aligned}$$

Si $f = f_1$ alors

$$Y(f_1) = X(f_1) - \frac{1}{K_t}\frac{\sin^2 \pi\, K_t (f_1 - f_0)}{\sin^2 \pi (f_1 - f_0)}$$

B.4 Diagonal Loading

Notons λ_x^c les valeurs propres fouillis et λ_x^{th} les valeurs propres bruit. De par la symétrie hermitienne de la matrice de covariance, il existe une matrice orthogonale $\mathbf{V}$ telle que :

$$\mathbf{R} = \mathbf{V} \begin{pmatrix} \lambda_1^c & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \lambda_2^c & 0 & & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & 0 & & \vdots \\ \vdots & & \ddots & \lambda_{n_0}^c & \ddots & & & \vdots \\ \vdots & & & \ddots & \lambda_1^{th} & \ddots & & \vdots \\ \vdots & & 0 & & \ddots & \lambda_2^{th} & \ddots & \vdots \\ \vdots & & & & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \lambda_{n_1}^{th} \end{pmatrix} \mathbf{V^H}$$

On voit alors que l'inverse de la matrice de covariance va être :

$$\mathbf{R^{-1}} = \mathbf{V} \begin{pmatrix} \lambda_1^{c\,-1} & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \lambda_2^{c-1} & 0 & & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & 0 & & \vdots \\ \vdots & & \ddots & \lambda_{n_0}^{c-1} & \ddots & & & \vdots \\ \vdots & & & \ddots & \lambda_1^{th\,-1} & \ddots & & \vdots \\ \vdots & & 0 & & \ddots & \lambda_2^{th\,-1} & \ddots & \vdots \\ \vdots & & & & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \lambda_{n_1}^{th\,-1} \end{pmatrix} \mathbf{V^H}$$

Étant donné que les valeurs propres du bruit thermique sont mal estimées, leur valeur peut être bien plus faible que la vraie valeur, ce qui non seulement peut poser des problèmes de conditionnement, mais peut aussi être très impactant sur le filtre, étant donné que l'inverse de cette valeur propre sera très grande. Ainsi, il existe une technique, appelée *Diagonal Loading*, ou « Chargement de diagonale », qui consiste à rajouter à la matrice de covariance une certaine fraction de la matrice de covariance du bruit thermique seul.

$$\mathbf{R_{DL}} = \mathbf{R} + \delta \mathbf{R_{th}}$$

où δ est le facteur de chargement. Cette opération a pour conséquence d'augmenter les faibles valeurs propres et permet d'éviter l'effet indésirable évoqué ci-dessus. Elle ajoute cependant un biais à la matrice de covariance. Nous voyons que si δ est très grand, la matrice de covariance tend vers $\delta \mathbf{R_{th}}$ et le traitement n'effectue alors plus de filtrage.

B.5 Autres définitions STAP

B.5.1 Clutter to Noise Ratio (CNR)

Le *Clutter to Noise Ratio* (CNR) représente le rapport fouillis (ou interférence) à bruit après filtrage. Il s'écrit

$$CNR = \frac{\mathbf{w^H R w}}{\mathbf{w^H \Gamma_{th} w}} - 1 \tag{B.4}$$

où $\mathbf{R}$ est la matrice de covariance estimée, $\mathbf{w^H}$ est le filtre STAP et $\mathbf{\Gamma_{th}}$ est la matrice de covariance thermique (du bruit) théorique. Si nous n'avons pas accès à $\mathbf{\Gamma_{th}}$, celle ci peut-être remplacée par la matrice de covariance du bruit estimée $\mathbf{R_{th}}$.

B.5.2 Adaptive Power Residue (APR)

La puissance estimée du résidu de signal parasite au niveau du vecteur de pointage, aussi appelée *Adaptive Power Residue* (APR), est définie par

$$APR = \mathbf{w^H R w}$$

Si $\mathbf{w^H} = \mathbf{s_s^H R^{-1}}$ alors

$$APR = E\{|(\mathbf{s_s^H R^{-1} x})^2|\} = \mathbf{w^H R w} = \mathbf{s_s^H R^{-1} s_s}$$

B.5.3 Blanchiment de la matrice de covariance

Si l'antenne utilisée est pondérée, les valeurs propres de la matrice de covariance le sont aussi. Pour appliquer des méthodes de détermination du nombre de sources, il est nécessaire de "blanchir" la matrice de covariance avant de procéder à sa décomposition en éléments propres. Si nous connaissons la vraie matrice de covariance du bruit $\mathbf{\Gamma_{th}}$ (qui est diagonale), soit la matrice $\mathbf{L}$

$$\mathbf{L} = \mathbf{\Gamma_{th}^{1/2}} \tag{B.5}$$

Si nous n'avons accès qu'à la matrice de bruit estimée $\mathbf{R_{th}}$, nous supprimons les termes croisés $\mathbf{R_{th}} = \mathbf{R_{th}} \circ \mathbf{I}$ afin de ne pas prendre en compte les éventuelles inter-corrélations entre capteurs. la matrice $\mathbf{L}$ est alors définie par

$$\mathbf{L} = (\mathbf{R_{th}} \circ \mathbf{I})^{\mathbf{1/2}} \tag{B.6}$$

La matrice de covariance blanchie $\mathbf{R'}$ est donc

$$\mathbf{R'} = \mathbf{L^{-1} R L^{-1}} \tag{B.7}$$

B.6 Matrices définies positives

B.6.1 Lemmes

Lemme 1. *Soit $GL(n)$ le groupe des matrices $n \times n$ inversibles. Chaque élément $\mathbf{X}$ donne une transformation congruente sur $\mathbb{M}_n$:*

$$\Gamma_{\mathbf{X}}(\mathbf{A}) = \mathbf{X}^*\mathbf{A}\mathbf{X}$$

$\Gamma_{\mathbf{X}} : \mathbf{X} \in GL(n)$ est un groupe de transformations sur $\mathbb{M}_n$.
Pour tout $\mathbf{X} \in GL(n)$ et pour chaque chemin différentiel γ

$$L(\Gamma_{\mathbf{X}} \circ \gamma) = L(\gamma) \tag{B.8}$$

B.6.2 Propositions

Proposition 1. *Soit $\mathbf{A}$ et $\mathbf{B}$ deux matrices qui commutent dans $\mathbb{P}_n$. Alors la fonction exponentielle relie le segment $[\log \mathbf{A}, \log \mathbf{B}]$ dans $\mathbb{H}_n$ à la géodésique $[\mathbf{A}, \mathbf{B}]$ dans $\mathbb{P}_n$. Dans ce cas*

$$\delta(\mathbf{A}, \mathbf{B}) = \| \log \mathbf{A} - \log \mathbf{B} \|$$

B.6.3 Demonstrations

Démonstration. Considérons le triangle dont les sommets sont $\mathbf{A}$, $\mathbf{B}$ et $\mathbf{C}$ et dont les cotés sont les géodésiques reliant les sommets. Soit $\mathbf{M_1} = \mathbf{A}\#\mathbf{B}$ le milieu du coté $[\mathbf{A}, \mathbf{B}]$ opposé au sommet $\mathbf{C}$ du triangle $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$. D'après l'Eq.(B.13) de l'Annexe B.6.4

$$\delta^2(\mathbf{M_1}, \mathbf{C}) \leq \frac{\delta^2(\mathbf{A}, \mathbf{C}) + \delta^2(\mathbf{B}, \mathbf{C})}{2} - \frac{\delta^2(\mathbf{A}, \mathbf{B})}{4} \tag{B.9}$$

Soit $\mathbf{M_2} = \mathbf{A}\#\mathbf{C}$. Dans le triangle $\{\mathbf{A}, \mathbf{M_1}, \mathbf{C}\}$ le point $\mathbf{M_2}$ est le milieu du coté $[\mathbf{A}, \mathbf{C}]$ opposé au sommet $\mathbf{M_1}$. Toujours d'après l'Eq.(B.13) de l'Annexe B.6.4, nous avons

$$\delta^2(\mathbf{M_1}, \mathbf{M_2}) \leq \frac{\delta^2(\mathbf{M_1}, \mathbf{C}) + \delta^2(\mathbf{M_1}, \mathbf{A})}{2} - \frac{\delta^2(\mathbf{A}, \mathbf{C})}{4} \tag{B.10}$$

En remplaçant la première inégalité et la deuxième, nous obtenons

$$\delta^2(\mathbf{M_1}, \mathbf{M_2}) \leq \frac{\delta^2(\mathbf{A}, \mathbf{C}) + \delta^2(\mathbf{B}, \mathbf{C})}{4} - \frac{\delta^2(\mathbf{A}, \mathbf{B})}{8} + \frac{\delta^2(\mathbf{M_1}, \mathbf{A})}{2} - \frac{\delta^2(\mathbf{A}, \mathbf{C})}{4} \tag{B.11}$$

Comme $\delta^2(\mathbf{M_1}, \mathbf{A}) = \delta^2(\mathbf{A}, \mathbf{B})/2$, le terme de droite de cette inégalité se reduit à $\delta^2(\mathbf{B}, \mathbf{C})/4$ ce qui prouve (4.22). Pour $0 \leq t \leq 1$ soit

$$\mathbf{A}\#_t\mathbf{B} = \mathbf{A}^{-1/2}(\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2})^t\mathbf{A}^{-1/2} \tag{B.12}$$

□

B.6.4 Loi du semi-parallélogramme

Proposition 1. *Si pour* $\mathbf{A}$, $\mathbf{B} \in \mathbb{P}_n$, *la matrice identité* $\mathbf{I}$ *se trouve sur la géodésique* $[\mathbf{A}, \mathbf{B}]$, *alors* $\mathbf{A}$ *et* $\mathbf{B}$ *commutent,* $[\mathbf{A}, \mathbf{B}]$ *est l'image isométrique sur la carte exponentielle d'un segment à travers* $\mathbf{O}$ *dans* $\mathbb{H}_n$, *et*

$$\log \mathbf{B} = -\frac{1-\xi}{\xi} \log \mathbf{A} \tag{B.13}$$

où $\xi = \delta(\mathbf{A}, \mathbf{I})/\delta(\mathbf{A}, \mathbf{B})$.

Preuve. d'après le **Théorème 1** du Chapitre 4.2.1, nous savons que

$$\mathbf{I} = \mathbf{A}^{-1/2}(\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2})^{\xi}\mathbf{A}^{-1/2}$$

où $\xi = \delta(\mathbf{A}, \mathbf{I})/\delta(\mathbf{A}, \mathbf{B})$, alors

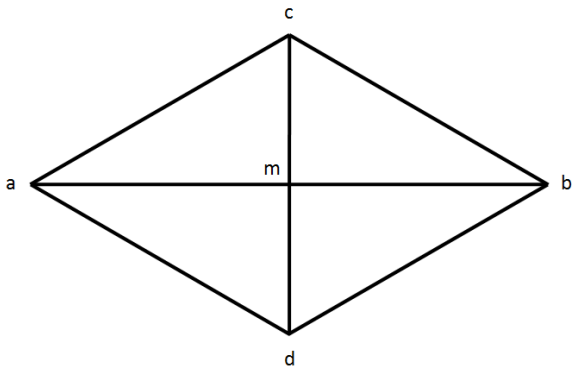
$$\mathbf{B} = \mathbf{A}^{1/2}\mathbf{A}^{-1/\xi}\mathbf{A}^{1/2} = \mathbf{A}^{-(1-\xi)/\xi}$$

donc $\mathbf{A}$ et $\mathbf{B}$ commutent et (B.13) est vérifiée. Une autre propriété essentielle de cette géométrie est la *loi du semi-parallélogramme* pour la métrique δ. Soit a et b deux points dans un espace de Hilbert $\mathcal{H}$ et soit $m = (a+b)/2$ leur milieu. Pour tout autre point c, considérons le parallélogramme dont une des diagonales est $[a, b]$ et l'autre est $[c, d]$. Ces deux diagonales se croisent en m et la loi du parallélogramme est l'égalité

$$\| a-b \|^2 + \| c-d \|^2 = 2(\| a-c \|^2 + \| b-c \|^2)$$

que nous pouvons ré-écrire

$$\| c-m \|^2 = \frac{\| a-c \|^2 + \| b-c \|^2}{2} - \frac{\| a-b \|^2}{4}$$



Dans la loi du semi-parallélogramme, cette dernière égalité est remplacée par une inégalité. □

Théorème (Loi du semi-parallélogramme). *Soit* $\mathbf{A}$ *et* $\mathbf{B}$ *deux points de* $\mathbb{P}_n$ *et soit* $\mathbf{M} = \mathbf{A}\#\mathbf{B}$ *le milieu de la géodésique* $[\mathbf{A}, \mathbf{B}]$. *Alors pour tout* $\mathbf{C}$ *dans* $\mathbb{P}_n$ *nous avons*

$$\delta^2(\mathbf{M}, \mathbf{C}) \leq \frac{\delta^2(\mathbf{A}, \mathbf{C}) + \delta^2(\mathbf{B}, \mathbf{C})}{2} - \frac{\delta^2(\mathbf{A}, \mathbf{B})}{4} \tag{B.14}$$

C

STAP sur GPU

C.1 Architecture ATI/AMD

Le concurrent de NVIDIA, a eu une approche différente dans le choix de l'architecture et dans l'implémentation d'une interface de programmation GP-GPU. ATI, devenu filiale d'AMD depuis 2006, a été le premier à proposer une API donnant accès au GPU pour le calcul. Cette API bas niveau appelée *Close to Metal* fut remplacée un an plus tard, en novembre 2007 par Brook+, une interface légèrement améliorée de Brook GPU développée par l'université de Stanford [93]. Cette API, aujourd'hui appelée AMD Stream, est beaucoup moins complète que l'interface CUDA de NVDIA et plus difficile à programmer ce qui explique le succès de NVIDIA sur ATI dans le calcul sur GPU. Au niveau de l'architecture de ses GPU, ATI a aussi eu une approche différente. Le GPU d'ATI abrite 20 grosses unités SIMD. Chacune de ces unités regroupe 16 processeurs vectoriels 5-way, soit en tout 1600 "cœurs". Il possède une puissance de calcul de 2720 GFlops en simple précision et 544 GFlops en double précision.

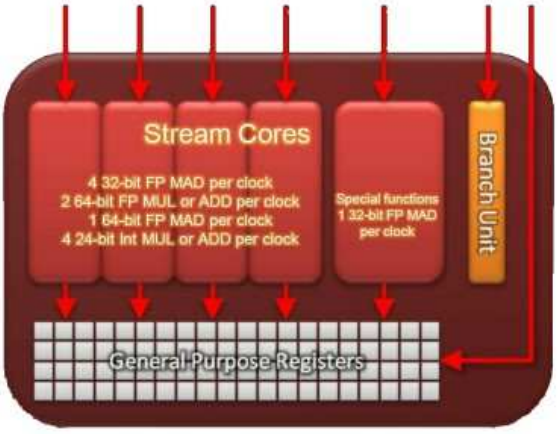


FIGURE C.1 – Schéma d'une unité de calcul du GPU AMD/ATI Cypress. Source : ATI

La structure de ces processeurs vec5 est de type 4+1. Chacune de ces 4 unités peut traiter soit une instruction simple (multiplication+addition) en FP32 par cycle, soit une addition FP64 en 2 cycles, soit une multiplication ou une multiplication+addition en 4 cycles. La dernière unité est distincte et peut traiter une opération FP32 par cycle ou une fonction spéciale (cos, sqrt...) en FP32 par cycle. Comme Fermi, Cypress supporte le nouveau standard IEEE 754-2008 qui est requis par DirectX 11. Ces processeurs vect5 sont, contrairement aux unités de type scalaire de NVIDIA, plus difficiles à exploiter. Le compilateur va devoir essayer de trouver 5 instructions indépendantes à exécuter en parallèle, ce n'est pas toujours possible. Ce type d'architecture est donc moins efficace mais en contrepartie on peut placer beaucoup plus d'unités de calcul sur une même surface. Ces 20 unités SIMD possèdent chacune une mémoire cache de 32 Ko. Nous trouvons aussi une mémoire cache globale partagée entre toutes les unités SIMD de 64 Ko. L'interface de programmation OpenCL [78] est en revanche en retard par rapport à NVIDIA CUDA, ce qui limite grandement l'utilisation des GPUs du fabricant.

C.2 Architecture d'un CPU moderne

Les processeurs généralistes compatibles PC, c'est à dire basés sur le jeu d'instruction x86 apparu en 1981 ont connu plusieurs évolutions depuis le début des années 1990. L'ajout de différentes technologies, poussé par un besoin croissant dans le domaine du multimédia, a permis à ces processeurs de s'imposer sur le segment du calcul haute performance face aux processeurs spécialisés dans le calcul scientifique basés sur le jeu d'instruction POWER [94] et leurs dérivés comme le processeur CELL [95]. Nous décrivons brièvement dans la suite l'architecture d'un CPU de type Intel Nehalem.

C.2.1 Les unités de calcul vectoriel

Depuis 2006 et l'arrivée des processeurs Core2, les CPU du constructeur Intel possèdent 3 unités de calculs sur les entiers (ALU), 3 unités sur les flottants (FP) et 3 unités SSE. Ces unités SSE sont en fait des unités vectorielles 128 bits. Chacune d'elle est capable d'effectuer en un seul cycle des opérations 128 bits, c'est-à-dire agissant simultanément sur quatre données 32 bits ou deux données 64 bits. En les utilisant, la puissance de calcul est donc multipliée par 2 en double précision et par 4 en simple précision.

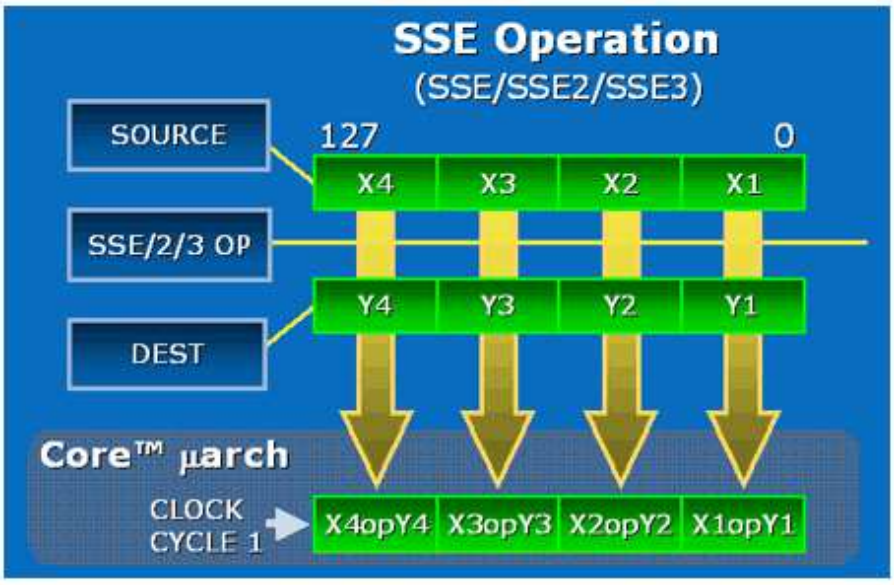


FIGURE C.2 – Structure d'une unité vectorielle SSE

Pour utiliser ces unités, nous pouvons soit utiliser des instructions spéciales dans les parties vectorisables du programme, soit utiliser la librairie Intel Math Kernel Library qui est une version optimisée des librairies de calculs libres (BLAS, FFTw, LAPACK), ou enfin activer l'auto-vectorisation dans les options de compilation. Cette dernière méthode est la plus simple mais aussi la moins efficace. En utilisant ces unités SSE, la puissance de calcul d'un cœur CPU est donné par la formule suivante :

```
P = 2 SIMD ops/cycle * 4 valeurs/op * fréquence d'horloge
```

C.2.2 OpenMP : programmation muticœurs

Après avoir atteint les limites de la montée en fréquence à cause de raisons physiques, les constructeurs de processeurs se sont tournés vers la parallélisation, mais à un plus haut niveau que les GPU. Un CPU possédant 4 cœurs, possède réellement 4 processeurs à part entière qui se partagent la même enveloppe physique et la même interface de transfert CPU-carte mère. La librairie OpenMP [96] est une des librairies [97][98] qui permet d'utiliser simplement les différents cœurs d'un CPU. OpenMP possède l'avantage d'être relativement simple à programmer (voir Figure C.3).

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
  int th_id, nthreads;
  #pragma omp parallel private(th_id)
  {
    th_id = omp_get_thread_num();
    printf("Hello World from thread %d\n", th_id);
    #pragma omp barrier
    if ( th_id == 0 ) {
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n",nthreads);
    }
  }
  return EXIT_SUCCESS;
}
```

FIGURE C.3 – Exemple simple de code OpenMP. Source : openmp.org

C.2.3 Hyperthreading

Enfin, les processeurs CPU possèdent généralement une fonction appelée Hyperthreading [99]. Elle permet d'émuler plusieurs cœurs logiques sur un seul cœur physique. Ainsi, chaque processeur physique (chaque cœur) possède comme 2 processeurs logiques associés. Ceci permet, dans le cas ou chaque tâche n'utiliserait pas un cœur à 100 % de ses capacités, de mieux remplir les unités de calculs en affectant 2 taches par cœur (Figure C.4). En revanche, dans le cas d'une bonne utilisation CPU, l'Hyperthreading ne donne aucun gain de performances. Pour tous nos tests, nous désactiverons l'Hyperthreading.

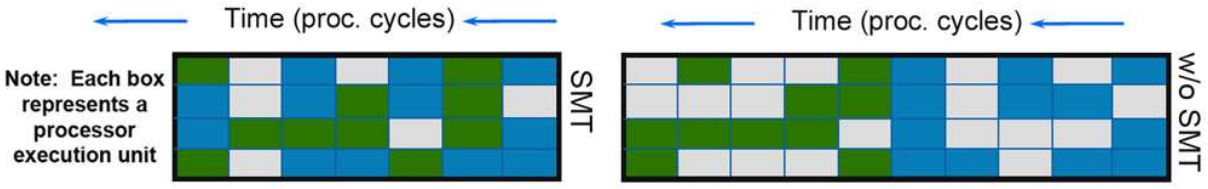


FIGURE C.4 – Execution de tâches avec (gauche) et sans (droite) Hyperthreading

C.3 Protocole de test : le PC de test

C.3.1 Caractéristiques générales

Carte mère : TYAN S7025
Processeurs CPU :
- Type : Intel Xeon X5570 (4 cœurs)
- Horloge CPU : 2,93 GHz
- Nombre : 2

Mémoire vive :
- Type : DDR3 (ECC)
- Horloge : 1333 MHz
- Taille : 6*4 Go = 24 Go

Cartes GPU :
- 1 * NVIDIA Quadro FX4800 (1,5 Go)
- 2 * NVIDIA Tesla C2050 (3 Go)

Système d'exploitation : Xubuntu 12.04 - 64 bits / Drivers NVIDIA 314.07 (CUDA 5.0)

FIGURE C.5 – Vue de l'intérieur du PC de test. Les deux CPUs (rectangles bleus) et les trois cartes GPU (rectangles verts) sont visibles

C.3.2 Les cartes GPU

Notre PC de test possède 3 cartes GPU. L'une des cartes, la NVIDIA Quadro 4800FX est une carte orientée graphisme professionnel. Bien que le GPU de cette carte soit tout à fait apte à faire du calcul scientifique, il est basé sur une architecture antérieure à l'architecture Fermi décrite dans le Chapitre 2, Partie 3. Deux d'entre elles, sont par contre des cartes NVIDIA Tesla C2050 orientées calcul. Nous avons vu aussi qu'un GPU d'architecture Fermi était composé de 16 multiprocesseurs SM. La puce se trouvant sur les cartes Tesla C2050 sont bien composées de 16 multiprocesseurs, mais seulement 14 sont activés, ce qui implique la présence de 448 "coeurs" et non pas 512. Nous pouvons calculer la puissance de calcul du GPU par la formule suivante :

```
P = 2 FMA ops/cycle * 448 coeurs * fréquence d'horloge
```

Les caractéristiques détaillées des deux cartes GPU sont les suivantes :

	NVIDIA Quadro FX4800	NVIDIA Tesla C2050
Puce	GT200	GF100
Capacités calcul	1.3	2.0
Coeurs CUDA	192	448
Fréquence coeurs CUDA	1200 MHz	1150 MHz
Puissance calcul (simple préc.)	462 GFlop/s	1030 GFlop/s
Puissance calcul (doublce préc.)	57.8 GFlop/s	515 GFlop/s
Mémoire vidéo	1.5 Go	3 Go
Interface mémoire	384-bit	384-bit
Fréquence mémoire	800 MHz	1.5 GHz
Bande passante mémoire	71.5 Go/s	144 Go/s
Interface	PCIe 2.0	PCIe 2.0
Conso. énergétique	150W	238 W

C.4 Taille des blocs et des threads : analyse sur GPU

Nous évaluons ici l'impact du nombre de *threads* et de *blocs* utilisés dans une fonction CUDA. Nous allons montrer qu'il n'est pas possible d'utiliser comme sur CPU des *threads* lourds sur GPU. Nous reprenons l'exemple de la multiplication de matrices décrit dans la Partie 3, Chapitre 3.2.1. Les fonctions CPU et GPU sont des fonctions non optimisées c'est à dire que nous programmons une fonction d'implémentation très simple pour CPU et pour GPU.

C.4.1 Test 1 : multiplication de matrices

Précisions des résultats

- CPU simple précision : $\max(\text{Cdiff}) = 5,38.10^{-6}$
- CPU double précision : $\max(\text{Cdiff}) = 6,57.10^{-12}$
- GPU simple précision : $\max(\text{Cdiff}) = 5,01.10^{-6}$
- GPU double précision : $\max(\text{Cdiff}) = 6,47.10^{-12}$

Temps de calcul

Nous mesurons le temps mis pour effectuer 180 multiplications de matrices 128×128 à la suite, 40 multiplications de matrices 1024×1024 à la suite et 1 multiplication de matrice 2048×2048. Nous affichons sur la Figure C.6 le gain de performances du GPU par rapport au CPU :

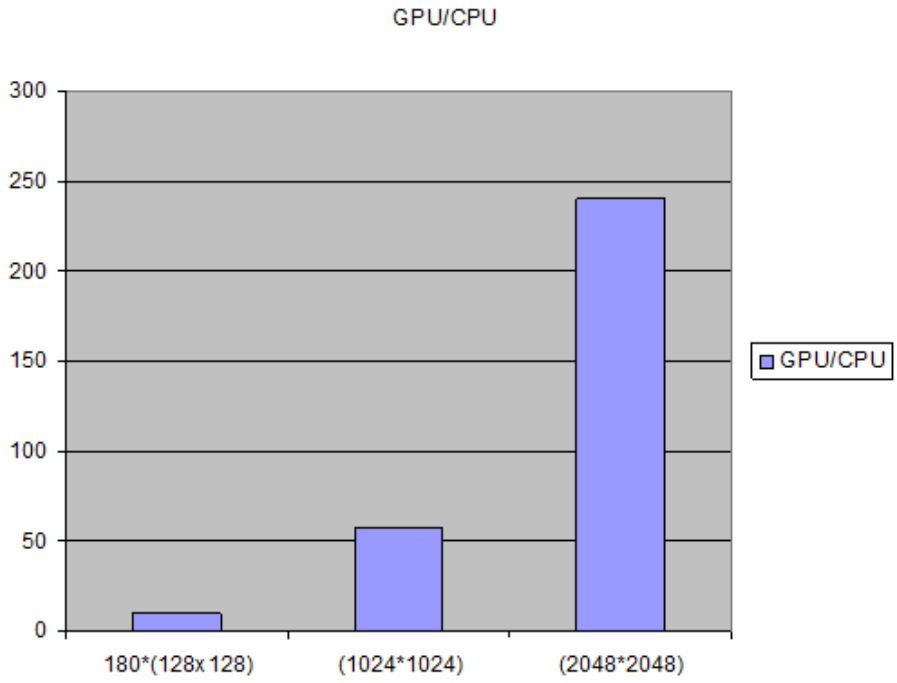


FIGURE C.6 – Ratio des performances GPU/CPU

Nous voyons que même si l'on traite plusieurs matrices à la suite, plus la taille des matrices est petite, plus le gain en performance qu'apporte le GPU est faible. Il faut donc arriver à traiter plusieurs matrices en parallèle sur GPU.

C.5 Répartition *thread* et *bloc*

Nous allons ici affecter 1 seul *thread* au calcul d'une matrice entière. Ce *thread* contiendra les $2N^3$ opérations de chaque matrice, soit ici 4,2 MFlop pour des matrices 128×128.

Temps de calcul

- 1 *bloc* de 180 *threads* : t = 5,6 s
- 180 *blocs* de 1 *thread* : t = 3,5 s

Avec la méthode utilisée précédemment, c'est à dire en affectant un *thread* par élément de matrice, le GPU affiche un temps de 0,124 s. Le CPU quant à lui met 1,17 s. Ici le GPU est donc moins performant que le CPU dans les deux cas. La première limitation semble venir des registres, qui ne peuvent contenir nos *threads*. NVIDIA Profiler nous indique sur la Figure C.7 que 16 registres sont utilisés par *thread* ce qui est le maximum avant la perte de rendement. Lors des essais précédents, sur une multiplication de matrice où chaque élément de matrice est associé à un *thread*, nous étions à 11 registres/*thread*.

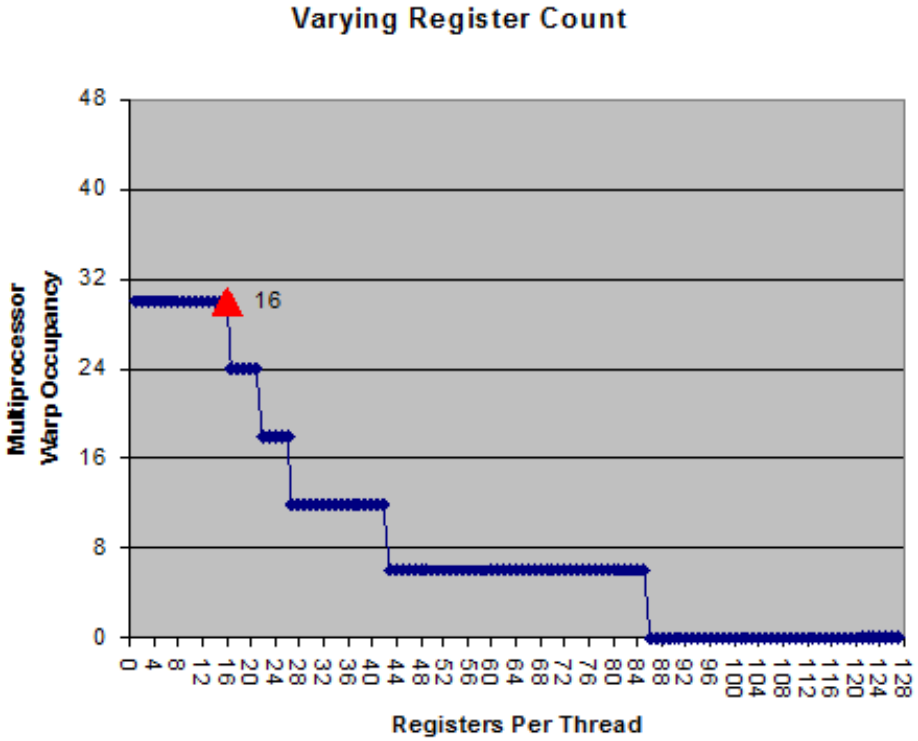


FIGURE C.7 – Rendement du multiprocesseur en fonction des registres/thread

Dans le cas 1 bloc de 180 *threads*, tout le calcul n'est traité que par un seul multiprocesseur. Notre carte de test, une NVIDIA Quadro 4800 possède 24 multiprocesseurs. Par contre, nous obtenons un bon rendement puisque d'après le calculateur de charge fourni par NVIDIA, avec 180 *threads*/bloc nous sommes à 30/32 soit 94% d'occupation du multiprocesseur (voir Figure C.8). A l'inverse en utilisant 180 *blocs* de 1 *thread*, nous perdons énormément en rendement par multiprocesseur (6,25%) mais nous utilisons tous les multiprocesseurs.

$$R1bloc = (1/24) * 0,94 = 0,039$$

$$R180blocs = (24/24) * (0,0625) = 0,0625 \longrightarrow R180/R1 = 1,6025$$

et

$$T1bloc = 5,6s$$

$$T180blocs = 3,5s \longrightarrow T1/T180 = 1,6$$

Nous retrouvons le rapport de temps obtenu lors de nos tests. Même si nous arrivons à trouver un compromis sur la taille du bloc, nous serons toujours limités par les registres. Cette approche n'est pas donc utilisable pour traiter plusieurs matrices en parallèle.

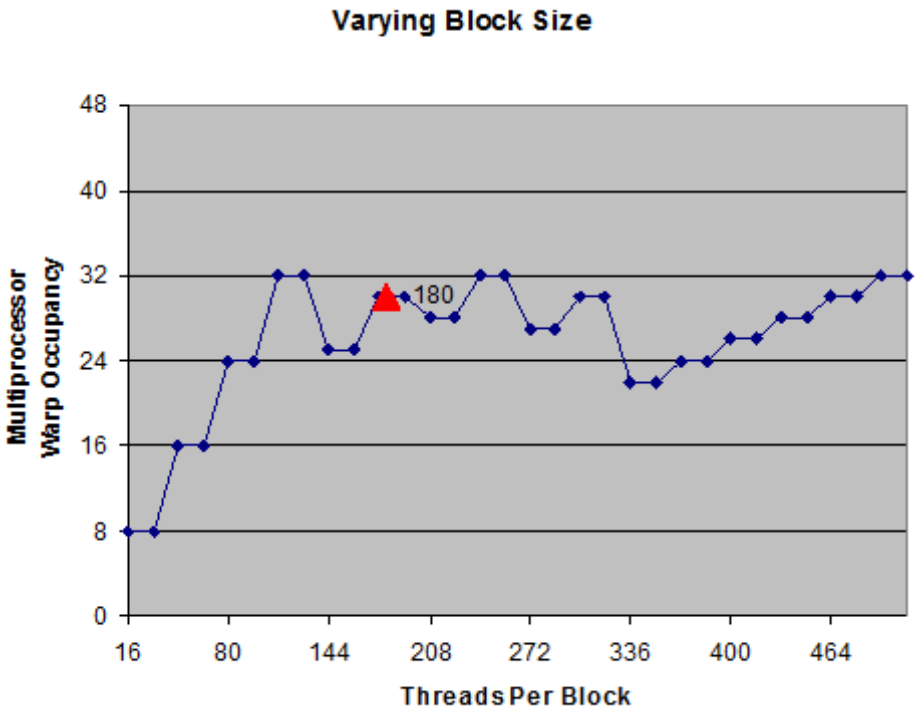


FIGURE C.8 – Charge du multiprocesseur en fonction de la taille du bloc

C.5.1 Plusieurs matrices , plusieurs *threads*

Enfin, nous allons effectuer les 180 multiplications de matrices en parallèle mais en affectant un *thread* au calcul d'un élément de chacun des termes de toutes les matrices de sorties. Pour cela, nous indexons les *threads* sur 3 dimensions, ligne, colonne et numéro de matrice. Les résultats ci-dessous montrent que cette approche, que nous pourrions appeler

Taille des blocs	Temps (secondes)
512 threads (16*16*2)	0,106
256 threads (16*16*1)	0,111
128 threads (8*8*2)	0,101

par *batch* est la plus efficace. C'est celle que nous utiliserons pour l'implémentation de la fonction d'estimation des matrices de covariance en STAP.

C.6 Codes

C.6.1 Matrixmulbatch de matrices sans mémoire partagée

```
float Cvalue = 0.0;
int row = blockIdx.y * blockDim.y + threadIdx.y;
int col = blockIdx.x * blockDim.x + threadIdx.x;
if(row > A.height || col > B.width) return;
for (int iboucle = 0; iboucle < A.width; ++iboucle)
Cvalue += (A.elements[row * A.width + iboucle])
* (B.elements[iboucle * B.width + col]);
C.elements[row * C.width + col] = Cvalue;
}
```

C.6.2 Matrixmulbatch avec mémoire partagée

```
// Blocs lignes et colonnes
int blockRow = blockIdx.y, blockCol = blockIdx.x;
// Chaque bloc calcule une sous-matrice Csub de C
Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
// Chaque thread calcule un element de Csub
  // et stocke le resultat dans Cvalue
float Cvalue = 0.0;
// Thread ligne et colonne dans Csub
int row = threadIdx.y, col = threadIdx.x;
// Boucle sur toutes les sous-matrices de A et de B
for (int m = 0; m < (A.width / BLOCK_SIZE); ++m)
// Transfert des portions de A et de B dans les sous-matrices
  // Asub et Bsub
Matrix Asub = GetSubMatrix(A, blockRow, m);
Matrix Bsub = GetSubMatrix(B, m, blockCol);
// Memoire partagee pour stocker Asub et Bsub
__shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
__shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
// Charge Asub et Bsub de la memoire global
  // vers la memoire partagee
As[row][col] = GetElement(Asub, row, col);
Bs[row][col] = GetElement(Bsub, row, col);
__syncthreads();
// Multiplication
for (int e = 0; e < BLOCK_SIZE; ++e)
Cvalue += As[row][e] * Bs[e][col];
__syncthreads();
```

C.6.3 Gemm avec flux

```
// Creation des streams
    streamArray = (cudaStream_t *)malloc
        (nb_batch * sizeof(cudaStream_t *));
    streamArray_batch =
           (cudaStream_t *)malloc(nb_batch * sizeof(cudaStream_t *));
    for (int iboucle = 0; iboucle < nb_batch ; iboucle++)
    {
            cudaError_t cudaErr =
                cudaStreamCreate(&streamArray[iboucle]);
            if (cudaErr != cudaSuccess)
            {
            fprintf(stderr, "!!!! cannot create stream\n");
                return EXIT_FAILURE;
            }
            streamArray_batch[iboucle] = 0;
    }

(...)

// Lancement des fonctions en parallele
for (iboucle = 0; iboucle < nb_batch ; iboucle++)
     {
          cublasSetStream(handle, streamArray[iboucle]);
          status = cublasCgemm(handle,transa,transb,M,
                        N,Kt,&cualpha,devPtrX[iboucle],M,
                        devPtrX[iboucle],N,&cubeta,
                        devPtrR[iboucle],N);

        if (status != CUBLAS_STATUS_SUCCESS)
        {
        cudaError_t cudaStatus = cudaGetLastError();
        fprintf(stderr, "!!!! GPU program execution error :
          cublasError=%d, cuda Error=%d,(%s)\n", status,
        cudaStatus,cudaGetErrorString(cudaStatus));
        return EXIT_FAILURE;
         }
     }
cudaThreadSynchronize();
```

C.7 Puissance de calcul mesurée

C.7.1 FFT

Mesures de la puissance de calcul obtenue pour effectuer une transformée de Fourier rapide (FFT) sur un totale de 16 millions de points en fonction de la taille du vecteur. Les données sont complexes et les calculs sont fait en simple précision (32bit).

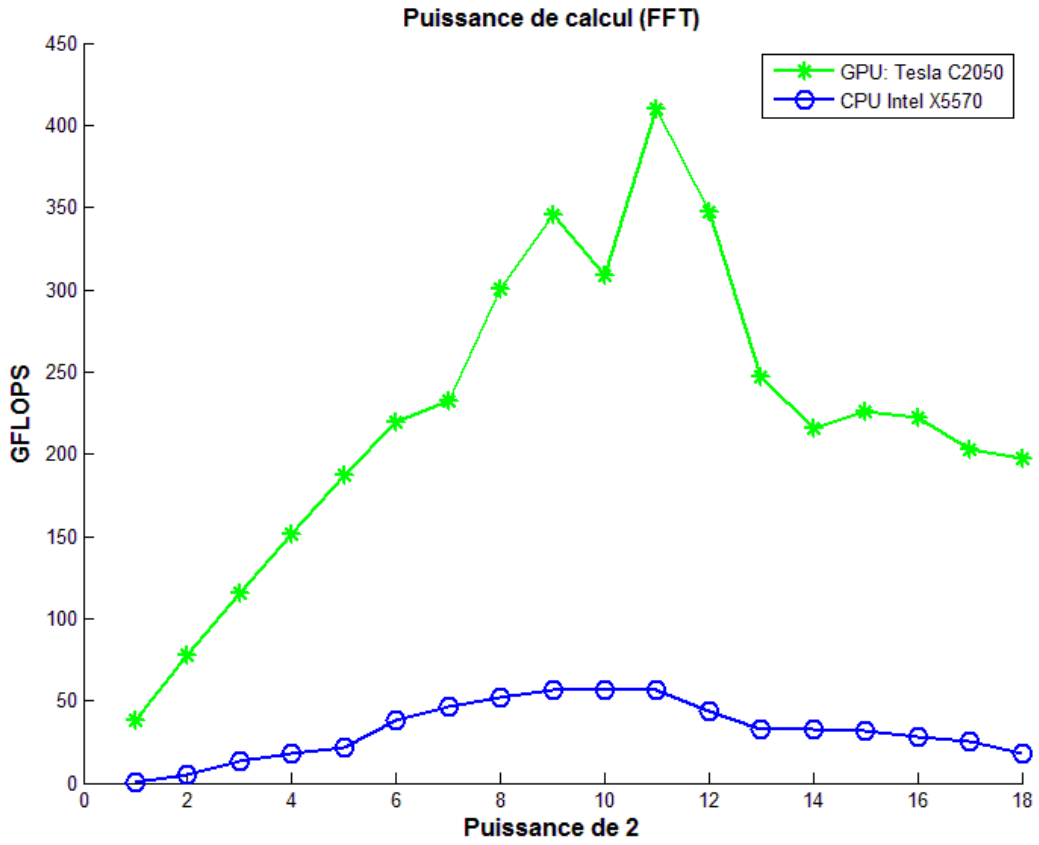


FIGURE C.9 – Puissance de calcul mesuré sur GPU Tesla C2050 et sur CPU X5570

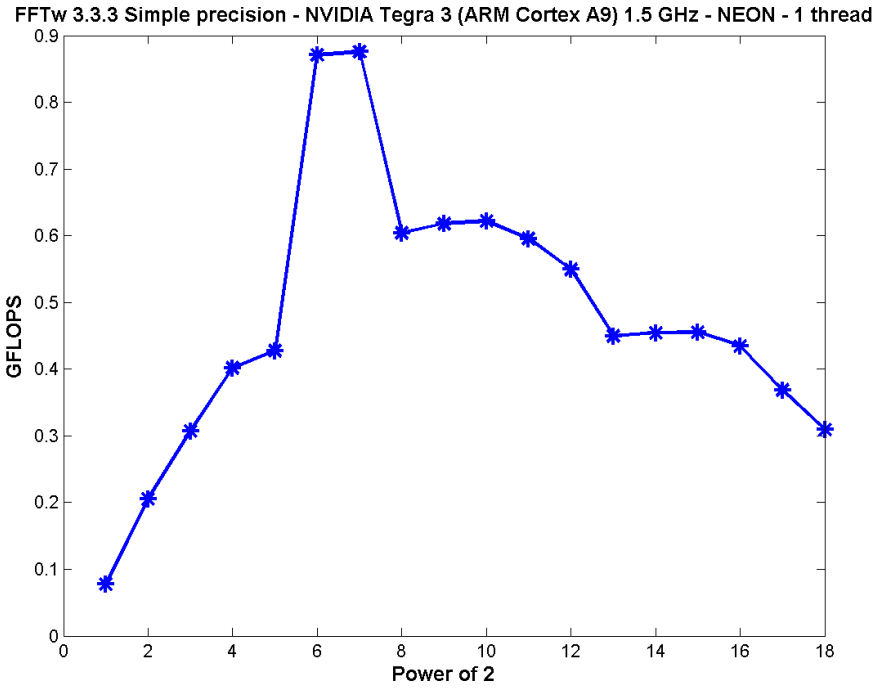


FIGURE C.10 – Puissance de calcul mesuré sur le CPU ARM de la carte CARMA

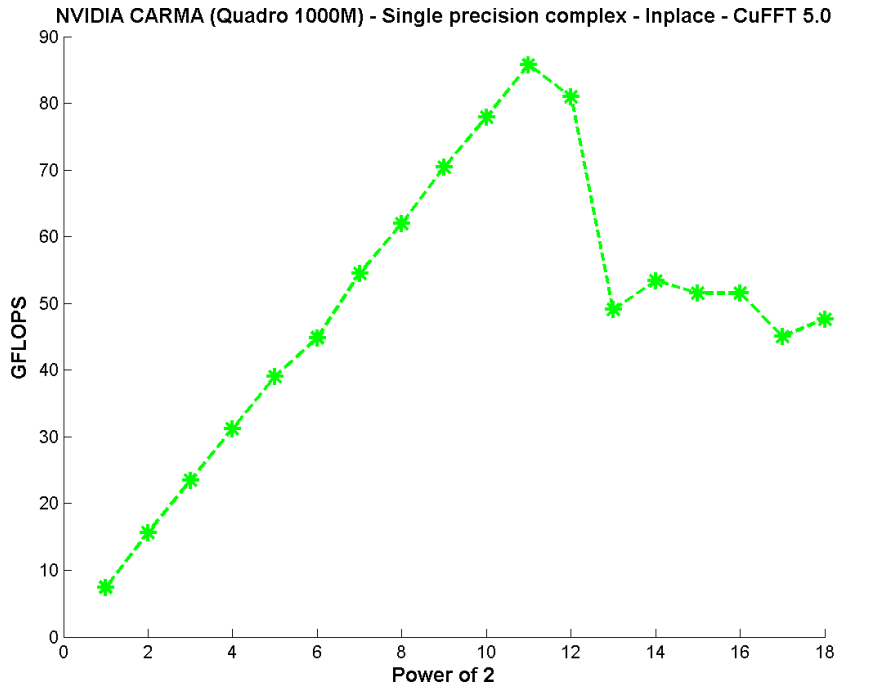


FIGURE C.11 – Puissance de calcul mesuré sur le GPU de la carte CARMA

Glossaire

Acronymes

ACE : *Adaptive Cosine Estimator*
AIC : *Akaike information criterion*
ALU : *Arithmetic Logic Unit*
AMF : *Adaptive Matched Filter*
APES : *Amplitude and Phase Estimation of a Sinusoide*
APR : *Adaptive Power Residue*
AVX : *Advanced Vector Extensions*
BFR : *Basse Fréquence de Récurrence*
CFAR : *Constant False Alarm Rate*
CNR : *Clutter to Noise Ratio*
CPU : *Central Processor Unit*
CSSP : *Constrained Sub-Space Projection*
CST : *Case Sous Test*
EC : *EigenCanceller*
EVD : *EigenValue Decomposition*
FAPI : *Fast Approximated Power Iteration*
FFT : *Fast Fourier Transform*
FIR : *Finite Impulse Response*
FPGA : *Field-Programmable Gate Array*
GMTI : *Generalized Likelihood Ratio Test*
GMTI : *Ground Moving Target Indication*
GPU : *Graphics Processor Unit*
MDL : *Minimum Description Length*
MFR : *Moyenne Fréquence de Récurrence*
MLED : *Maximum Likelihood Estimation Detector*
PRF : *Pulse Repetition Frequency*
PRI : *Pulse Repetition Interval*
SCM : *Sample Covariance Matrix*
SER : *Surface Équivalente Radar*
SIRV : *Spherically Invariant Random Vectors*
SMI : *Sample Matrix Inversion*
SSE : *Streaming SIMD Extensions*
STAP : *Space-time adaptive Processing*
SVD : *Singular Value Decomposition*
SWaP : *Size, Weight and Power*
TFAC : *Taux de Fausse Alarme Constant*

Notations

$\mathbf{a^H}$: *transconjugué ou conjugué hermitien du vecteur* $\mathbf{a}$. *Est égal à* $(\mathbf{a}^*)^T$ *ou* $(\mathbf{a^T})^*$
$\otimes$: *produit de Kronecker*
$\| \, . \, \|_F$: *Norme de Frobenius*
$\mathbf{\Gamma}$: *Matrice de covariance des interférences théorique*
$\mathbf{R}$: *Matrice de covariance des interférences estimée*
$\mathbf{\Gamma_{th}}$: *Matrice de covariance du bruit thermique théorique*
$\mathbf{R_{th}}$: *Matrice de covariance du bruit thermique estimée*
f_p : *Fréquence porteuse*
P_d : *Probabilité de détection*
P_{fa} : *Probabilité de fausse alarme*
$\mathbb{M}_n$: *Ensemble des matrices complexes de taille* $n \times n$
$\mathbb{H}_n$: *Ensemble des matrices Hermitiennes de taille* $n \times n$
$\mathbb{P}_n$: *Ensemble des matrices Hermitiennes strictement positives de taille* $n \times n$
$\gamma(t)$: *Géodésique de paramètre* t
$GL(n)$: *Groupe des matrices inversibles de taille* $n \times n$

Bibliographie

[1] Laurent Savy and Francois Le Chevalier. *Traitements spatio-temporels adaptatifs en radar*. Ed. Techniques Ingénieur, 2009.

[2] Charles E Muehe and Melvin Labitt. Displaced-phase-center antenna technique. *Lincoln Laboratory Journal*, 12(2) :281–296, 2000.

[3] Peter G Richardson. Space-time adaptive processing for manoeuvring airborne radar. 1998.

[4] Gregory Hellbourg, Tual Trainini, Rodolphe Weber, Eric Moreau, Cecile Capdessus, and AJ Boonstrd. Rfi subspace estimation techniques for new generation radio telescopes. In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, pages 200–204. IEEE, 2012.

[5] Gregory Hellbourg, Rodolphe Weber, Cécile Capdessus, Albert-Jan Boonstra, Rym Feliachi, et al. Approches spatiales cyclostationnaires pour le traitement des interférences en radioastronomie. *les radiotélescopes du futur, Technologies et avancées scientifiques*, 2011.

[6] Guillaume Carrie, François Vincent, Thierry Deloues, David Pietin, Alain Renard, and Franck Letestu. Optimal stap algorithms to gnss receivers. 2006.

[7] Remi Letestu, Magali Le Garff-Tavernier, Dominique Vaur, Michel Ticchioni, Fanny Baran-Marszak, Frederic Davi, Veronique Salaun, Franck Genevieve, Virginie Eclache, Dina Naguib, et al. Analysis of b-cll with discordant zap-70 expression and igvh mutational status. *Blood*, 106(11) :1194–1194, 2005.

[8] Stéphanie BIDON. Introduction au stap : 2. modèle des signaux et principe du filtrage. *TS. Traitement du signal*, 28(1-2) :35–55, 2011.

[9] Jacques DARRICAU. Radars : Principes et éléments de base. *Techniques de l'ingénieur. Télécoms*, (E6650) :E6650–1, 1996.

[10] James Ward. Space-time adaptive processing for airborne radar. 1998.

[11] W.L. Melvin. A STAP overview. *IEEE Aerospace And Electronic Systems Magazine*, 19(1) :19–35, 2004.

[12] R. Klemm. *Principles of Space-time adaptive processing*. The Institution of Electrical Engineers (IEE), 2002.

[13] Stéphanie Bidon, Olivier Besson, and Jean-Yves Tourneret. Synthèse des traitements stap pour la détection en environnement hétérogène. *TS. Traitement du signal*, 28(1-2) :81–112, 2011.

[14] Y.L. Kim, S.U. Pillai, and J.R. Guerci. Optimal loading factor for minimal sample support space-time adaptive radar. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 4, pages 2505 – 2508 vol.4, may 1998.

[15] M. Zatman. Properties of hung-turner projections and their relationship to the eigencanceller. In *Signals, Systems and Computers, 1996. Conference Record of the 30th Asilomar Conference on*.

[16] Scott D Berger and Byron M Welsh. Selecting a reduced-rank transformation for stap-a direct form perspective. *Aerospace and Electronic Systems, IEEE Transactions on*, 35(2) :722–729, 1999.

[17] J Scott Goldstein, Irving S Reed, and Louis L Scharf. A multistage representation of the wiener filter based on orthogonal projections. *Information Theory, IEEE Transactions on*, 44(7) :2943–2959, 1998.

[18] L.E. Brennan and I.S. Reed. Theory of adaptive radar. *IEEE Transactions On Aerospace And Electronic Systems*, AES-9(2) :237–252, 1973.

[19] Geordi K Borsari. Mitigating effects on stap processing caused by an inclined array. In *Radar Conference, 1998. RADARCON 98. Proceedings of the 1998 IEEE*, pages 135–140. IEEE, 1998.

[20] DANIEL J Rabideau and AO Steinhardt. Improved adaptive clutter cancellation through data-adaptive training. *Aerospace and Electronic Systems, IEEE Transactions on*, 35(3) :879–891, 1999.

[21] Stephen M Kogon and Michael A Zatman. Stap adaptive weight training using phase and power selection criteria. In *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*, volume 1, pages 98–102. IEEE, 2001.

[22] Pinyuen Chen, William L Melvin, and Michael C Wicks. Screening among multivariate normal data. *Journal of Multivariate Analysis*, 69(1) :10–29, 1999.

[23] K Gerlach, SD Blunt, and ML Picciolo. Robust adaptive matched filtering using the fracta algorithm. *Aerospace and Electronic Systems, IEEE Transactions on*, 40(3) :929–945, 2004.

[24] Louis L Scharf and L Tood McWhorter. Adaptive matched subspace detectors and adaptive coherence estimators. In *Signals, Systems and Computers, 1996. Conference Record of the Thirtieth Asilomar Conference on*, pages 1114–1117. IEEE, 1996.

[25] Olivier Besson. Detection in the presence of surprise or undernulled interference. *Signal Processing Letters, IEEE*, 14(5) :352–354, 2007.

[26] Stéphanie Bidon, Olivier Besson, and J-Y Tourneret. The adaptive coherence estimator is the generalized likelihood ratio test for a class of heterogeneous environments. *Signal Processing Letters, IEEE*, 15 :281–284, 2008.

[27] Jean-Philippe Ovarlez, Frédéric Pascal, Philippe Forster, Guillaume Ginolhac, Mélanie Mahot, et al. Traitement stap et modélisation sirv : Robustesse et persymétrie. *Traitement du signal*, 28(1-2) :113–142, 2011.

[28] Melanie Mahot. *Estimation robuste de la matrice de covariance en traitement du signal*. PhD thesis, Ecole doctorale de Sciences Pratiques, Ecole Normale Superieure de Cachan, 2012.

[29] Peter Mandl and Udeepta Bordoloi. General-purpose graphics processing units deliver new capabilities to the embedded market, 2011.

[30] M.A. Clark, P.C. La Plante, and L.J. Greenhill. Accelerating Radio Astronomy Cross-Correlation with Graphics Processing Units. 2011.

[31] Marcos S. Savy L. Molinie J-Ph. Degurse J-F, Dugrosprez B. Architecture gpu pour un radar de surveillance spatial et pour radars aéroportés. In *GRESTI Conference 2013*.

[32] VK Veligatla, P Labropoulos, and LVE Koopmans. Adaptive beam-forming for radio astronomy on gpu.

[33] Bin Liu, Kaizhi Wang, Xingzhao Liu, and Wenxian Yu. An efficient sar processor based on gpu via cuda. In *Image and Signal Processing, 2009. CISP'09. 2nd International Congress on*, pages 1–5. IEEE, 2009.

[34] Timo Balz and Uwe Stilla. Hybrid gpu-based single-and double-bounce sar simulation. *Geoscience and Remote Sensing, IEEE Transactions on*, 47(10) :3519–3529, 2009.

[35] Hubert MJ Cantalloube and Carole E Nahum. Airborne sar-efficient signal processing for very high resolution. 2013.

[36] Jimmy Pettersson and Ian Wainwright. Radar signal processing with graphics processors (gpus). *Master's thesis, Uppsala University*, 2010.

[37] John R Humphrey, Daniel K Price, Kyle E Spagnoli, Aaron L Paolini, and Eric J Kelmelis. Cula : hybrid gpu accelerated linear algebra routines. In *SPIE Defense, Security, and Sensing*, pages 770502–770502. International Society for Optics and Photonics, 2010.

[38] Michael Roeder, Nolan Davis, Jeremy Furtek, Dennis Braunreiter, and Dennis Healy. Gpu implementations for fast factorizations of stap covariance matrices. In *Optical Engineering+ Applications*, pages 707403–707403. International Society for Optics and Photonics, 2008.

[39] E. Aboutanios and B. Mulgrew. A STAP algorithm for radar target detection in heterogeneous environments. In *Statistical Signal Processing, 2005 IEEE/SP 13th Workshop on*.

[40] Li H. Stoica P. and J Li. A new derivation of the apes filter. *IEEE Signal Processing Letters*, pages 205–206, 1999.

[41] E. Aboutanios and B. Mulgrew. Evaluation of the single and two data set STAP detection algorithms using measured data. *2007 IEEE International Geoscience and Remote Sensing Symposium*, pages 494–498, 2007.

[42] Dan C Youla and Heywood Webb. Image restoration by the method of convex projections : Part 1 theory. *Medical Imaging, IEEE Transactions on*, 1(2) :81–94, 1982.

[43] SU Pillai, JR Guerci, and SR Pillai. Performance analysis of data sample reduction techniques for stap. In *Phased Array Systems and Technology, 2003. IEEE International Symposium on*, pages 565–570. IEEE, 2003.

[44] Henry Stark. *Image recovery : theory and application*. Access Online via Elsevier, 1987.

[45] Lawrence E Brennan and LS Reed. Theory of adaptive radar. *Aerospace and Electronic Systems, IEEE Transactions on*, (2) :237–252, 1973.

[46] P. Forster. Generalized rectification of cross spectral matrices for arrays of arbitrary geometry. *Signal Processing, IEEE Transactions on*, 49(5) :972–978, 2001.

[47] A. Haimovich. The eigencanceler : adaptive radar by eigenanalysis methods. *IEEE Transactions on Aerospace and Electronic Systems*, 1996.

[48] R. Badeau, B. David, and G. Richard. Fast approximated power iteration subspace tracking. *Signal Processing, IEEE Transactions on*, 53(8) :2931 – 2941, aug. 2005.

[49] S. Beau and S. Marcos. Range dependent clutter rejection using range-recursive space-time adaptive processing (stap) algorithms. *Signal Processing*, 90(1) :57–68, 2010.

[50] Hirotugu Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6) :716–723, 1974.

[51] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5) :465–471, 1978.

[52] Vladimir Alexandrovich Marchenko and Leonid Andreevich Pastur. Distribution of eigenvalues for some sets of random matrices. *Matematicheskii Sbornik*, 114(4) :507–536, 1967.

[53] Jack W Silverstein. Eigenvalues and eigenvectors of large dimensional sample covariance matrices. *Contemporary Mathematics*, 50 :153–159, 1986.

[54] G.M. Herbert. A new projection based algorithm for low sidelobe pattern synthesis in adaptive arrays. In *Radar 97 (Conf. Publ. No. 449)*, pages 396 –400, oct 1997.

[55] D.H. Brandwood and J.D. Baker. Stabilisation of adaptive array patterns using signal space projection. In *Antennas and Propagation, 1989. ICAP 89., Sixth International Conference on (Conf. Publ. No.301)*, pages 289 –294 vol.1, apr 1989.

[56] Elias Aboutanios and Bernard Mulgrew. Hybrid detection approach for STAP in heterogeneous clutter. *Aerospace and Electronic Systems, IEEE Transactions on*, 46(3) :1021 –1033, july 2010.

[57] J. Angulo. Mathematical morphology for matrix-valued images. In *MATRIX INFORMATION GEOMETRIES*.

[58] A Aubry, A De Maio, L Pallotta, A Farina, and C Fantacci. Median matrices and geometric barycenters for training data selection. In *Radar Symposium (IRS), 2013 14th International*, volume 1, pages 331–336. IEEE, 2013.

[59] *Positive Definite Matrices*, chapter 6. Princeton University Press, 2006.

[60] W Pusz and S Lech Woronowicz. Functional calculus for sesquilinear forms and the purification map. *Reports on Mathematical Physics*, 8(2) :159–170, 1975.

[61] *Leçons sur la geometrie des espaces de Riemann.* Deuxieme edition, revue et augmentee, 1946.

[62] Dario A Bini and Bruno Iannazzo. Computing the karcher mean of symmetric positive definite matrices. *Linear Algebra and its Applications*, 2011.

[63] Dario Andrea Bini and Bruno Iannazzo. A note on computing matrix geometric means. *Advances in Computational Mathematics*, 35(2-4) :175–192, 2011.

[64] Carl Ludwig Siegel. Symplectic geometry. *American Journal of Mathematics*, 65(1) :1–86, 1943.

[65] Frédéric Barbaresco. Information geometry of covariance matrix : cartan-siegel homogeneous bounded domains, mostow/berger fibration and frechet median. In *Matrix Information Geometry*, pages 199–255. Springer, 2013.

[66] H. Karcher. Riemannian center of mass and mollifier smoothing. *Comm. Pure Appl. Math*, 1977.

[67] B. Balaji and F. Barbaresco. Application of riemannian mean of covariance matrices to space-time adaptive processing. In *Radar Conference (EuRAD), 2012 9th European*, pages 50–53, 2012.

[68] Le Yang, Marc Arnaudon, and Frédéric Barbaresco. Geometry of covariance matrices and computation of median. In *AIP Conference Proceedings*, volume 1305, page 479, 2011.

[69] Frédéric BARBARESCO. Géométrie différentielle des matrices de covariance et espaces métriques à courbure négative. In *XXIIe colloque GRETSI (traitement du signal et des images), Dijon (FRA), 8-11 septembre 2009*. GRETSI, Groupe d'Etudes du Traitement du Signal et des Images, 2009.

[70] M Arakawa. Computational workloads for commonly used signal processing kernels. *Lincoln Laboratory*, 2006.

[71] Herb Sutter and James Larus. Software and the concurrency revolution. *Queue*, 3(7) :54–62, 2005.

[72] Ian Buck. Gpu computing : Programming a massively parallel processor. In *Code Generation and Optimization, 2007. CGO'07. International Symposium on*, pages 17–17. IEEE, 2007.

[73] Craig M Wittenbrink, Emmett Kilgariff, and Arjun Prabhu. Fermi gf100 gpu architecture. *Micro, IEEE*, 31(2) :50–59, 2011.

[74] *NVIDIA's Next Generation CUDA Compute Architecture :Fermi.*

[75] Nathan Whitehead and Alex Fit-Florea. Precision & performance : Floating point and ieee 754 compliance for nvidia gpus. *rn (A+ B)*, 21 :1–1874919424, 2011.

[76] AdvancedMicroDevices. Ati radeon hd 5870 - gpu.

[77] Intel. Corei7-975. http ://ark.intel.com/products/37153.

[78] John E Stone, David Gohara, and Guochun Shi. Opencl : A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3) :66, 2010.

[79] Jason Sanders and Edward Kandrot. *CUDA by example : an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[80] Jack W Davidson and Sanjay Jinturkar. Memory access coalescing : a technique for eliminating redundant memory accesses. In *ACM SIGPLAN Notices*, volume 29, pages 186–195. ACM, 1994.

[81] Bo Kågström, Per Ling, and Charles Van Loan. Gemm-based level 3 blas : High-performance model implementations and performance evaluation benchmark. *ACM Transactions on Mathematical Software (TOMS)*, 24(3) :268–302, 1998.

[82] Chetan Jhurani and Paul Mullowney. A gemm interface and implementation on nvidia gpus for multiple small matrices. *arXiv preprint arXiv :1304.7053*, 2013.

[83] David B Kirk and W Hwu Wen-mei. *Programming massively parallel processors : a hands-on approach*. Morgan Kaufmann, 2010.

[84] D. Pastre. Méthode de gauss-jordan. calcul de l'inverse d'une matrice. In *Université René Descartes, Méthodes Numériques*.

[85] Dave Jaggar et al. Arm architecture and systems. *IEEE micro*, 17(4) :9–11, 1997.

[86] San Jose GPU Tech Conference 2012. Carma : Cuda on arm architecture, developments in power-efficient computing. In *GPU Tech Conference 2012, San Jose*.

[87] Vasily Volkov. Better performance at lower occupancy. In *Proceedings of the GPU Technology Conference, GTC*, volume 10, 2010.

[88] H Cantalloube and P Dubois-Fernandez. Airborne x-band sar imaging with 10 cm resolution : technical challenge and preliminary results. *IEE Proceedings-Radar, Sonar and Navigation*, 153(2) :163–176, 2006.

[89] J.-L. Milin, S. Moore, W. Burger, P.-Y. Triboulloy, M. Royden, and J. Gerster. Amsar - a european success story in aesa radar. *Aerospace and Electronic Systems Magazine, IEEE*, 25(2) :21 –28, feb. 2010.

[90] Stéphanie BIDON, Marc MONTECOT, and Laurent SAVY. Introduction au stap : 3. les données du club stap. *TS. Traitement du signal*, 28(1-2) :57–79, 2011.

[91] S. Busson. See : un simulateur pour la conception d'un systeme d'observation spatiale par radar. *SEE*, 1988.

[92] P. Dubois-Fernandez, O. Ruault du Plessis, D. Le Coz, J. Dupas, B. Vaizan, X. Dupuis, H. Cantalloube, C. Coulombeix, C. Titin-Schnaider, P. Dreuillet, J.M. Boutry, J.P. Canny, L. Kaisersmertz, J. Peyret, P. Martineau, M. Chanteclerc, L. Pastore, and J.P. Bruyant. The ONERA RAMSES SAR system. In *Geoscience and Remote Sensing Symposium, 2002. IGARSS '02. 2002 IEEE International*, volume 3, pages 1723–1725 vol.3, 2002.

[93] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for gpus : stream computing on graphics hardware. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 777–786. ACM, 2004.

[94] Shlomo Weiss and James E Smith. *IBM Power and PowerPC*. Morgan Kaufmann Publishers Inc., 1994.

[95] H Peter Hofstee. Power efficient processor architecture and the cell processor. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 258–262. IEEE, 2005.

[96] Leonardo Dagum and Ramesh Menon. Openmp : an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1) :46–55, 1998.

[97] Chikara Sakamoto, Teruki Miyazaki, Masayuki Kuwayama, Keizo Saisho, and Akira Fukuda. Design and implementation of a parallel pthread library (ppl) with parallelism and portability. *Systems and Computers in Japan*, 29(2) :28–35, 1998.

[98] Shirley Browne, Jack Dongarra, and Kevin London. Review of performance analysis tools for mpi parallel programs. *NHSE Review*, 3(1) :241–248, 1998.

[99] David Koufaty and Deborah T Marr. Hyperthreading technology in the netburst microarchitecture. *Micro, IEEE*, 23(2) :56–65, 2003.

Résumé

Les traitements spatio-temporels adaptatifs (STAP) sont des traitements qui exploitent conjointement les deux dimensions spatiale et temporelle des signaux reçus sur un réseau d'antennes, contrairement au traitement d'antenne classique qui n'exploite que la dimension spatiale, pour leur filtrage. Ces traitements sont particulièrement intéressants dans le cadre du filtrage des échos reçus par un radar aéroporté en provenance du sol pour lesquels il existe un lien direct entre direction d'arrivée et fréquence Doppler. Cependant, si les principes des traitements STAP sont maintenant bien acquis, leur mise en œuvre pratique face à un environnement réel se heurte à des points durs non encore résolus dans le contexte du radar opérationnel.

Le premier verrou, adressé par la thèse dans une première phase, est d'ordre théorique, et consiste en la définition de procédures d'estimation de la matrice de covariance du fouillis sur la base d'une sélection des données d'apprentissage représentatives, dans un contexte à la fois de fouillis non homogène et de densité parfois importante des cibles d'intérêts.

Le second verrou est d'ordre technologique, et réside dans l'implémentation physique des algorithmes, lié à la grande charge de calcul nécessaire. Ce point, crucial en aéroporté, est exploré par la thèse dans une deuxième phase, avec l'analyse de la faisabilité d'une implémentation sur GPU des étapes les plus lourdes d'un algorithme de traitement STAP.

Mots-clés: Radar, STAP, traitement du signal adaptatif, GPU, détection, fouillis hétérogène

Abstract

Space-time adaptive processing (STAP) is a processing that makes use of both the spatial and the temporal dimensions of the received signals by an antenna array, whereas conventional antenna processing only exploits the spatial dimension to perform filtering. These processing are very powerful to remove ground echoes received by airborne radars, where there is a direct relation between the arrival angle and the Doppler frequency. However, if the principles of STAP processing are now well understood, their performances are limited when facing practical situations.

The first part of this thesis, is theoretical, and consists of defining effective procedures to estimate the covariance matrix of the clutter using a representative selection of training data, in a context of both non-homogeneous clutter and sometimes high density of targets.

The second point studied in this thesis is technological, and lies in the physical implementation of the selected algorithms, because of their high computational workload requirement. This is a key point in airborne operations, and is explored by the thesis in a second phase, with the analysis of the feasibility of implementation on GPU of the heaviest stages of a STAP processing.

Keywords: Radar, STAP, adaptive signal processing, GPU, detection, heterogeneous clutter